Proceedings of the Fall 2006 Workshop of the HPI Research School on Service-Oriented Systems Engineering

Benjamin Hagedorn, Michael Schöbel, Matthias Uflacker, Flavius Copaciu, Nikola Milanovic

Technische Berichte Nr. 18
des Hasso-Plattner-Instituts
für Softwaresystemtechnik an der Universität Potsdam

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

Nr. 18

Proceedings of the Fall 2006 Workshop of the HPI Research School on Service-Oriented Systems Engineering

Benjamin Hagedorn, Michael Schöbel, Matthias Uflacker,
Flavius Copaciu, Nikola Milanovic

Potsdam 2007

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.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:	Benjamin Hagedorn, Michael Schöbel, Matthias Uflacker, Flavius Copaciu, Nikola Milanovic
Email:	{benjamin.hagedorn; michael.schoebel; matthias.uflacker; flavius.copaciu; nicola.milanovic}@.hpi.uni-potsdam.de
Vertrieb:	Universitätsverlag Potsdam Am Neuen Palais 10 14469 Potsdam Fon +49 (0) 331 977 4517 Fax +49 (0) 331 977 4625 e-mail: ubpub@uni-potsdamde http://info.ub.uni-potsdam.de/verlag.htm
Druck:	allprintmedia gmbH Blomberger Weg 6a 13437 Berlin email: info@allprint-media.de

Heft Nr 18 (2007)
ISBN 978-939469-58-2
ISSN 1613-5652

Content

1. **Design and Composition of 3D Geoinformation Services**
   Benjamin Hagedorn

2. **Operating System Abstractions for Service-Based Systems**
   Michael Schöbel

3. **A Task-oriented Approach to User-centered Design of Service-Based Enterprise Applications**
   Matthias Uflacker

4. **A Framework for Adaptive Transport in Service-Oriented Systems based on Performance Prediction**
   Flavius Copaciu

5. **Asynchronicity and Loose Coupling in Service-Oriented Architectures**
   Nikola Milanovic

Design and Composition of 3D Geoinformation Services

Benjamin Hagedorn

benjamin.hagedorn@hpi.uni-potsdam.de

Forced by Google Earth the topic of geoinformation has become mainstream. As 80 per cent of all digital information carries a spatial reference, there is a great opportunity for using geographic information as a basis for gaining insight into these data, for more efficient processing, and for developing new applications on the basis of such information. Especially 3D geoinformation gets more and more important for a variety of application domains. For enabling the interoperable access and processing of geoinformation, standards organizations as the Open Geospatial Consortium have specified different services and data description languages. This work gives an introduction to the field of 3D geoinformation services and points out open questions that arise from using geoservices. The focus is on the visualization aspects and the consequences for service design and integration with existing and future geoservice-based systems.

1 Introduction

Geoinformation is used in a lot of contexts and is a factor for economic success. To facilitate geoinformation, to integrate it in different applications and to see things that were not obvious before, means to create value. 3D geoinformation supports this creation of new insights into data and processes. For activating the capabilities of geoinformation it must be collected, processed, and distributed efficiently for being used effectively. Service-based architectures are a concept for achieving this integration of geoinformation and geographic information services offer the technology to implement them. In modern times geographic information becomes main stream for a variety of applications in very different domains. The main issue of this development is to activate the value that is contained in geographic information, combine it with processes and other information, and so to generate added value.

The underlying vision of this work is the development of 3D geoinformation services with the objective of activating the inherent potential of geoinformation by its integration in 3D geovirtual environments.

This paper takes a look at the topic of geoservices and how they can be used to create higher-level functionality. Chapter 2 discusses geoinformation and points on the usage of internet mapping and geodata infrastructures as a possibility for distributing geodata and correlated functionality. Chapter 3 is about geoservices themselves and how they can be assembled, about geoservices standards, and open research questions in this field. Chapter 4 describes the current project work for the OGC Web Services Initiative, Phase 4, which is about interoperability between different geodata and between different vendors. In the conclusion an outlook on further research is given.

2 Geoinformation and Geovisualization

2.1 Geoinformation

Geodata is data about concrete or abstract objects, terrain shapes and infrastructure elements on the earth surface. The common property of geodata is their spatial reference. It is estimated that about 80% of all information has a spatial reference, which can be used as a basis for processing and visualizing such data. Considering the context of meaning and usage, geodata becomes geoinformation that has spatial properties (geometry and topology) and semantic properties (semantics and thematic attributes):

- *Geometry* includes information about the shape and position of geoobjects. Geometry can be given, e.g., as points, lines, or polygons.
- *Topology* describes the spatial relationship between different geoobjects, e.g., adjacency or containedness.
- *Semantics* describes the meaning of a geoobject in a concrete context, e.g., if it is a road, runway, or building.
- *Thematic data* describes non-spatial properties of geoobjects, e.g., building material or speed limits for roads, number of occupants or household income, or temporal attributes as year of construction.

The semantic classification is essential for the proper application of geoinformation. Hierarchical classification allows abstraction and enables user- and task-oriented processing of geoinformation.

Geoinformation is collected by a variety of organizations. Public authorities are committed by law to collect and maintain geodata. On the other hand, commercial organizations gather geodata for their own purposes or for others. This results in a huge and distributed amount of heterogeneous geoinformation which is not inherently interoperable because of varying purposes, coverages, different measurement methods, classification schemas, storage formats, spatial precision or timeliness.

Geoinformation is an essential system component for an increasing number of applications and systems as for facility management, logistics, security, telecommunication, disaster management, location-based services, real estate portals as well as entertainment and education products.

2.2 From GIS to GDI

2.2.1 GIS and Internet-Mapping

Geographic Information Systems (GIS) are software systems for collecting, storing, managing, analyzing, and visualizing geospatial data. For a long time, GIS have been monolithic systems, which allowed the users to work efficiently with the data. Geoinformation is distributed over these systems and extra effort is necessary for reaching a consistent data state. Additionally, multitudes of applications need extra effort for administration.

Today, large network bandwidth and distributed application platforms as they are provided by application servers and browser-based front-ends allow a distributed GIS which consists of a GIS server and appropriate GIS clients. According to the functionalities and the separation of concerns of server and client, Fitzke et al. (1997) propose five categories of web-based geoinformation systems, see Table 1.

Table 1: GIS categories and their provided GIS functionality.

Category of web-based GIS	Provided functionalities
Geodata server	Data management
Map server	Data management, visualization
Online retrieval system	Data management, visualization, retrieval
Online GIS	Data management, visualization, retrieval, GIS analysis
GIS function server	Visualization, retrieval, GIS analysis

Internet-Mapping and real Web-GIS (which must include GIS analysis functionality) help to integrate the geoinformation that is further distributed in a company. It is no longer stored locally but in a central database. This prevents data redundancies and inconsistency. Furthermore a central GIS server and simple web based clients reduce the effort for software administration.

2.2.2 Geodata Infrastructures

Worldwide, there are efforts to establish so-called geodata infrastructures (GDI). The term GDI is not a well defined one. It can be described to conclude all entities that serve in the provision, transportation, and processing of geoinformation. The primary goal of these infrastructures is to make the collected but widely distributed geoinformation available for re-use in administration and economy. In Germany, there are several regional GDI projects, as in Berlin, Brandenburg or North Rhine-Westphalia. Above those, Germany installed a national GDI project which itself is covered by the European GDI project INSPIRE. Geoinformation services play an important role in these geodata infrastructures as they are often used to enable the access to geodata. As geoinformation services often base on standards they support the interoperability of geoinformation and between different participants in the geodata infrastructure.

In the context of GDIs, there are still a number of open questions about legal issues, the organization of this infrastructure, its architecture, and the geodata that shall be provided – e.g., who is allowed to participate in the infrastructure, how to participate, or how much it would be.

2.3 3D Geovisualization

Visualization "is concerned with exploring data and information in such a way as to gain understanding and insight into the data [and] to promote a deeper level of understanding of the data under investigation and to foster new insight into the underlying processes […]." (Brodlie et al., 1992)

So, geovisualization is the visual representation of data with a special reference, respectively. Not only spatial objects and processes but also those objects and processes that can be transformed into spatial geometry are target of geovisualization.

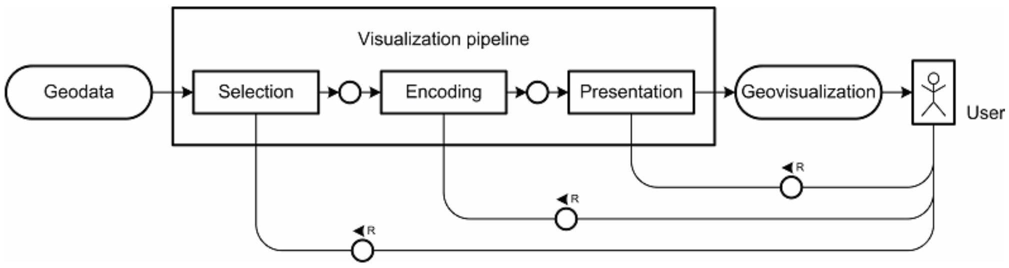


Figure 1: Geovisualization pipeline. (Based on Spence 2001.)

According to Spence (2001), the process of visualizing geoinformation can be organized in the geovisualization pipeline which contains the selection, encoding, and presentation stages (Figure 1):

- *Selection stage*: The geodata that shall be visualized are reformatted, integrated, processed and selected accordingly to the task to fulfill. Result is a model of the geodata.
- *Encoding stage*: The selected geodata is transferred into a geovirtual model. Thereby the geodata is mapped on a computer graphics representation (model objects and attributes).
- *Presentation stage*: Images of the computer graphics model are synthesized into the final geovisualization that is perceptible by a human user.

The geovisualization pipeline provides the interaction of the user with the visualization system which is quite demonstrative for a GIS. E.g., it enables the user to load other geoinformation, to influence the visual encoding (e.g., colors or map symbols), or to switch between different views or to move the virtual camera.

As traditional GIS applications dealt with 2D geoinformation, 3D geoinformation became more and more important for a variety of applications in the last years and has found its way into GIS.

3D geovirtual environments (GeoVE) are capable of representing urban spatial and geo-referenced data, including terrain models, building models, vegetation models as well as models of roads and transportation systems. A GeoVE can be used in the obvious way for representing city objects of the real world, but it also can be utilized as a platform for the integration of abstract geoinformation, such as noise level information or visibility information.

Compared to 2D geovisualization, 3D geovisualization deals with large amounts of geometry data or texture data and needs special computer graphics techniques and intensively uses the computer graphics hardware. An other important issue is the enablement of interaction with the GeoVE, such as navigation which is the most important interaction technique as it enables the user to move through the GeoVE for gathering the contained geoinformation.

On the other hand well known methods in the field of 2D geovisualization, e.g., cartographically generalization, can not be applied to 3D geovisualization easily.

3 Geoinformation Services

3.1 Service-oriented Computing

After object-oriented and component-oriented programming, service-based computing is called to be a new programming paradigm using possibly distributed and network-connected services as more abstract functional entities on a more architectural view on a software system. Software architectures considering this programming paradigm are called service-oriented architectures (SOA) and are pushed to the markets by a variety of vendors, currently. From a management viewpoint, a SOA is a management concept which targets on a flexible IT infrastructure which is aligned to the business goals and can be easily adapted to the changing business environment. From a technical viewpoint a SOA is a software architecture concept which bases on the usage of software services.

ISO 19119 (2005) defines a service as a collection of operations which are accessible through an interface and allow a service consumer to evoke a behavior of value at the provider offering the service. For enabling the service consumer to find and utilize the service functionality, the service interface must be described and published in a standardized way. Therefore, the service repository is defined as a third participant in a SOA. Figure 2 shows these SOA participants and the described publish-find-bind interaction pattern.

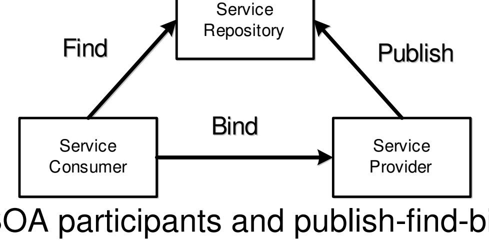


Figure 2: SOA participants and publish-find-bind pattern.

The service consumer uses the described service interface to invoke the service's functionality – the concrete service implementation is hidden. This leads to a loose coupling of the application entities in a SOA, increases their reusability and supports interoperability.

The standardization of interfaces and exchange formats allows to combine services in ways that are not predefined and to assemble the service functionality for achieving larger tasks. Service composition is described by service orchestration or service choreography – this is the service combination from a higher system top view or from the local service view, respectively. The consequence of service composition is that the involved service providers may also act as consumers of the functionality that is provided by other services.

XML web services are one possible service implementation for building a SOA. They use technologies as Web Service Description Language (WSDL) for describing the service (e.g., how to invoke), UDDI for publishing the service, SOAP as a message exchange protocol over HTTP as transport protocol.

Representational State Transfer (REST) is another concept for implementing service-based systems. It bases on the idea of resources which are defined by unified resource locators (URL). REST supports a binding to HTTP only. The HTTP operations GET, PUT, POST, and DELETE are used for interacting with the resources, e.g., for retrieving representations of a resource, creating new resources, or deleting them.

3.2 Geoservices

In the context of geoinformation, services offer the possibility to provide widely distributed geoinformation in an interoperable manner and facilitate the reuse of data and functionality in a variety of applications. Geoservices support the various forms of web-based GIS but also are important components of a GDI. So, geoservices offer the possibility to integrate geoinformation into business process, e.g., for spatial data mining or decision support (Andrienko 1999, 2005).

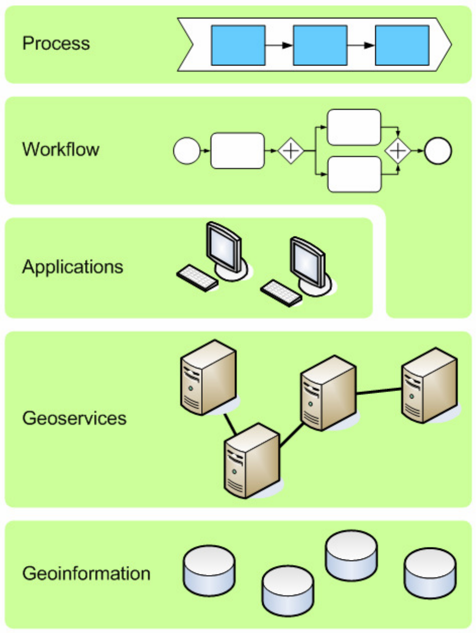


Figure 3: Geoservices in the context of SOA views.

Figure 3 denotes the integration of geoinformation and geoservices in the context of business integration. It is made up of the following entities:

- *Business process* which is an abstract description of a set of steps for reaching a business goal: It is the "means by which one or more activities are accomplished in operating business practices." (ebXML 2001)
- *Workflow* which is the "automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules." (Workflow Management Coalition 1999)
- *Applications* which allow a user to be part of the process execution in a special workflow step.
- *Geoservices* which might be dedicated for the use in a workflow in an automated manner or might be consumed by application clients for human interaction.
- *Geoinformation* which are distributed over a multitude of sources and can be accessed via services.

Geoservices can be categorized according to their functionality. Typical geoservice types are:

- *Geodata services* which provide geoinformation, e.g., road network information.
- *Portrayal services* which provide visualizations, e.g., maps.
- *Processing services* which, e.g., provide transformations of geodata or may synthesize new geoinformation.

Especially for geodata services, the service repository should allow the service consumer to search for specific geoinformation (e.g., roads or building information) in a specific spatial area which could be defined by a bounding box. Service composition is another major topic for geoservices.

3.2.1 Interoperability

The utilization of geoservices advances the interoperability of different geoinformation-sources and the possibility to integrate geoinformation in a new context for gaining new insights. Interoperability is a challenge in discovering, accessing, and using geoinformation. For solving these challenges geoservice-based systems offer meta information, standardized service interfaces and standardized data models.

On a conceptual level Bishr (1998) distinguishes the following three categories of interoperability problems in GIS:

- *Semantic heterogeneity* characterizes the different understanding of features of the real world. E.g., a road network is a network structure for routing but a set of laminar objects.
- *Schema heterogeneity* characterizes differences in the structure of the model. E.g., geoobjects are modeled on a class level in one system and as attributes in another system.
- *Syntactical heterogeneity* characterizes differences in the exchange format on the one hand and in the geometric representation on the other hand – e.g., the storage of road network as vector data vs. the storage as raster data.

Gröger and Kolbe (2003) point out additional interoperability problems concerning geometry and topology of single geoobjects as they can be caused by different spatial reference systems, or errors in measurement, digitizing or generalization. These problems of geometrical and topological interoperability can be further subdivided into geometrical and topological inconsistencies, problems arising from multi-scalar representations, and problems caused by geodata infrastructures as data inconsistencies.

3.2.2 Geoservice Chaining

In the field of geoinformation services, service composition is described by the term service chaining. Alameh (2003) offers three architecture patterns for service chaining that base on chaining patterns that are provided by ISO 19119 (2005), see Figure 4. The main difference in this classification is the issue of control about the service chain workflow and the visibility of the involved services for the user:

- *Client-coordinated service chaining*: The user has full knowledge about the participating services and controls the workflow of the service chain.
- *Workflow-managed service chaining*: The user knows about the involved services but invokes a workflow management service which controls the workflow of the service chain.
- *Static chaining using aggregate services*: The user does not know about the participating services. They are hidden by an aggregate service which controls the workflow.

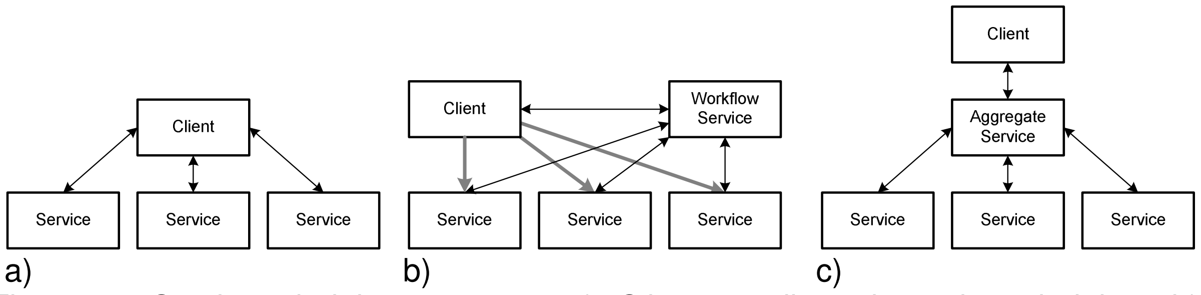


Figure 4: Service chaining patterns: a) Client-coordinated service chaining; b) Workflow-managed service chaining (grey: client "knows" about services); c) Static chaining using aggregate services.

These composition patterns correspond to service orchestration, as a central manager (client, workflow service, or aggregate service) has the responsibility for the service chain execution. Furthermore, the patterns can be adapted in the sense, that one service forwards the response directly to another service. Such service nesting is more efficient as possible large data must be transferred only once.

Especially for geoservices, composition is an important issue for the integration of different geoinformation. Chainable geoservices are, e.g., coordinate transformation services, routing services, or generalization services.

E.g., one possible geoservice chaining might result from the integration of different geodata sets into one visualization: a) Two different geodata services provide road network information in different coordinate reference system. b) A processing services has to transform both geodata sets into the same coordinate reference system. c) A portrayal service can use this geodata for synthesizing a map visualization.

3.3 Standards for Geoservices

3.3.1 OGC Web Services

Several organizations are active in the field of geographic standards. Those are, e.g., the International Organization for Standardization (ISO), the Open Geospatial Consortium (OGC), the Federal Geographic Data Committee (FGDC), or the European Committee for Standardization (CEN).

The OGC is a non-profit, international standards organization which develops and promotes standards for geospatial services. The OGC has defined several implementation specifications for services and for data exchange formats. Furthermore a row of discussion papers is waiting for becoming an OGC specification. The following list gives an overview on some of the well known, often used, and interesting OGC specifications and discussion papers.

Geography Markup Language (GML) GML is an encoding specification for geodata in XML for storing, exchanging, and processing geographic information.

CityGML Currently, CityGML has the status of a discussion paper in the OGC standardization process. It is a GML-based format for the description of 3D geovirtual environments. This means geometry and topology but also the semantics of city model objects. This semantics is not provided by the abstract GML specification. Such semantics are, e.g., building, wall, door, water body, etc.

Web Map Service (WMS) The WMS defines the creation and display of map-like views of distributed data. The WMS supports the following operations:

- GetMap requests a map for a defined bounding rectangle with specified information layers included, and in a specified graphical style.
- GetFeatureInfo is an optional operation. It provides additional information about the geographical features that in a map at a special pixel position.

Web Feature Service (WFS) The WFS provides an interface to data stored in GML. It allows a service consumer to retrieve and manipulate these geoinformation. Among others the WFS provides the following operations:

- GetFeature is the operation for retrieving feature instances.
- DescribeFeatureType is the functionality to describe the structure of every feature that can be retrieved from GetFeature.
- WFS might offer transactions, which are composed of requests for data modification: Create, Update, Delete.

Web Coverage Service (WCS) The WCS is capable of providing geospatial data as "coverages" which are raster data sets. Different from WMS this are not image data per default but raw geographical data that can be interpreted by the service consumer. Important operations of the interface are:

- DescribeCoverage describes the coverages that are named by parameter.
- GetCoverage enables the access to coverage data. Parameters are coverage size, coverage format and interpolation.

Web Catalogue Service (WCS) Web Catalogues serve for discovering OGC web services and retrieving service metadata.

Web Terrain Service (WTS) The WTS is a perspective view service – this is, it provides 3-dimensional views of geovirtual models which may include terrain, buildings, vegetation, etc.

Web 3D Service (W3DS) The W3DS is a service for 3D geodata. It provides 3D a scene graphs which is a computer graphics model that must be rendered for retrieving imagery that can be perceived by the human.

Filter Encoding The Filter Encoding specifies an encoding for filter expression in XML. Filter encodings are used as functions for obtaining a subset from a set of objects. Filter encodings can be used in variety of OGC web services for reducing the result set, e.g. when specified together with the WFS operation GetFeature.

3.3.2 The Google-Way

Google Earth is an application for accessing and visualizing different types of geoinformation, as orthophotos, terrain models, city models, or infrastructure elements as roads or airports. It allows to include own paths or models into the scene and to fly directly to a town or landmark. Considerable building models can be easily created with Google SketchUp, an easy to use sketching tool for building construction. These models can be imported and positioned into Google Earth by the usage of the simple data transfer format KML which is an easy description for building sites.

Google Maps is another geodata view client. The browser-based viewer provides detailed ortophotos of the earth surface and offers an alternative map view. Additionally, Google offers the Google Maps API which enables programmers to access the Google Maps service and to integrate maps into their own homepage. The Google Maps API offers additional functionality as positioning arbitrary map symbols, add own information layer or even to integrate map layers from an OGC web map service. A lot of people used the possibilities of the Google Maps API and created their own geovisualization, called mashups. An example is the georeferenced map display of apartments to rent enriched with additional information – see Figure 5.

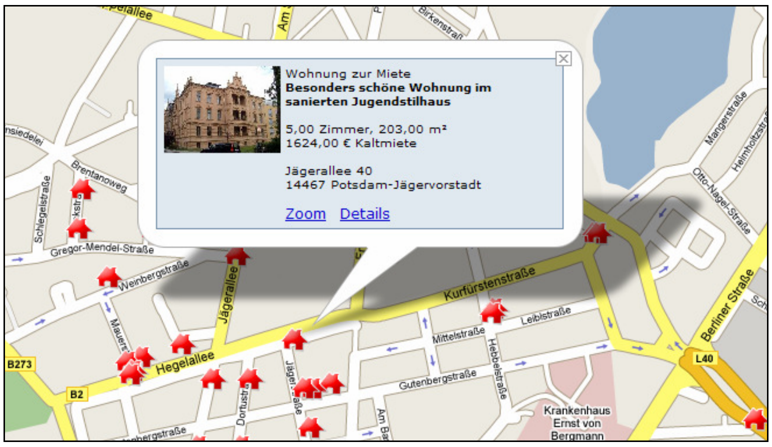


Figure 5: Mashup of Google Maps and apartments for rent. (mapits.de)

Google Earth and Google Maps are successful products that enable everybody to use geovisualization and so have set up a quasi standard in the field of service geoinformation processing and visualization. The mashups with Google Maps generate an added-value to the users. Both are excellent examples how to integrate and facilitate geoinformation for gaining insight.

3.4 The OGC Portrayal Model

The geovisualization pipeline that was introduced in 2.3 corresponds to the OGC portrayal model which is shown in Figure 6. This model is annotated with some possible participants covering different functionalities of the pipeline. (Schmidt et al. 2003).

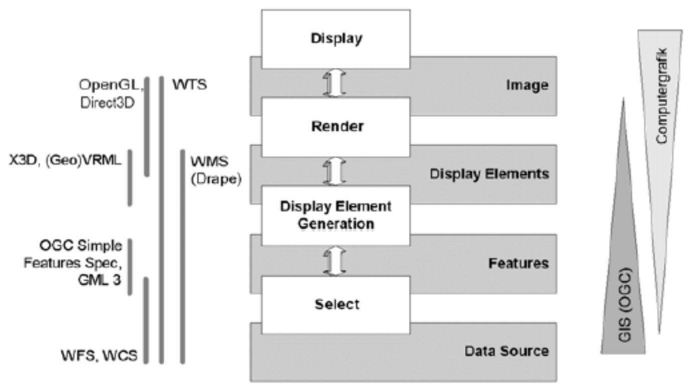


Figure 6: OGC portrayal model; modified by Schmidt (2003).

The OGC defines portrayal as information presentation to the human – portrayal elements may be either images or display elements. Corresponding to this, the OGC defines providing portrayal services and consuming "application services" as part of their OWS service framework (Percivall 2003). Application services may be either application servers, or application clients.

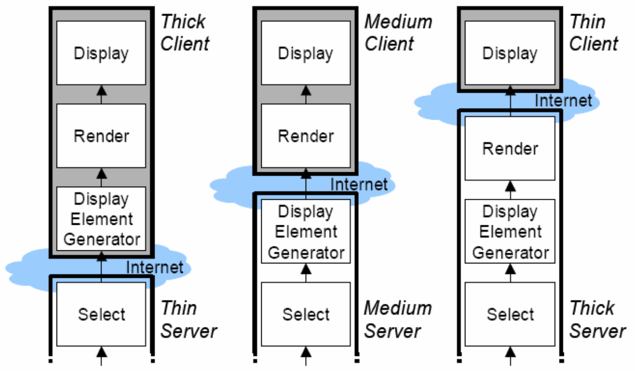


Figure 7: Partition concepts of the visualization pipeline in service-based systems. (Altmaier and Kolbe 2003)

The application of geoservices for geovisualization raises the question of the separation of rendering concerns between service provider (portrayal service) and service consumer (application service), this means between server and clients. The OGC Portrayal model allows three partitions which are illustrated by Altmaier and Kolbe (2003) in Figure 7 for client/server architectures:

- Thick Client / Thin Server: The client request selected data from the server and performs the remaining steps of the geovisualization pipeline. This requires appropriate computer graphics capabilities of the client. A possible geoservice participating in this scenario is a WFS.
- Medium Client / Medium Server: The service provides computer graphical representations (e.g., a VRML scenegraph) to the client which has to synthesize images. Again the client needs computer graphics capabilities for rendering images. In this scenario the style of the resulting visualization is more in concern of the server. The W3DS is a possible participant in this scenario.
- Thin Client / Thick Server: All the visualization steps are performed by the server. The server defines the final visualization and the client must only provide capabilities for displaying the visualization to the end-user.

3.5 Open Questions

Figure 8 illustrates the topics that are relevant when addressing 3D geoinformation visualization via service-based systems. Vital parts of such systems are 3D geoservice providers and 3D geoservice consumer.

The service interface describes the service capabilities to the service consumer. The implementation of these capabilities is basing on the columns of modeling, rendering, and interaction:

- *Modeling* addresses the structure and organization of complex geoinformation.
- *Rendering* addresses the mapping of this geoinformation onto computer graphics elements relevant for geovirtual environments and the synthesis of images of this GeoVE. These images support the end-user in understanding the geoinformation and in getting insight into the underlying data.

- *Interaction* addresses the issue of navigation inside the GeoVE but also the manipulation of the presented objects and their underlying geoinformation and the analysis of these data which might lead to the generation of new geoinformation.

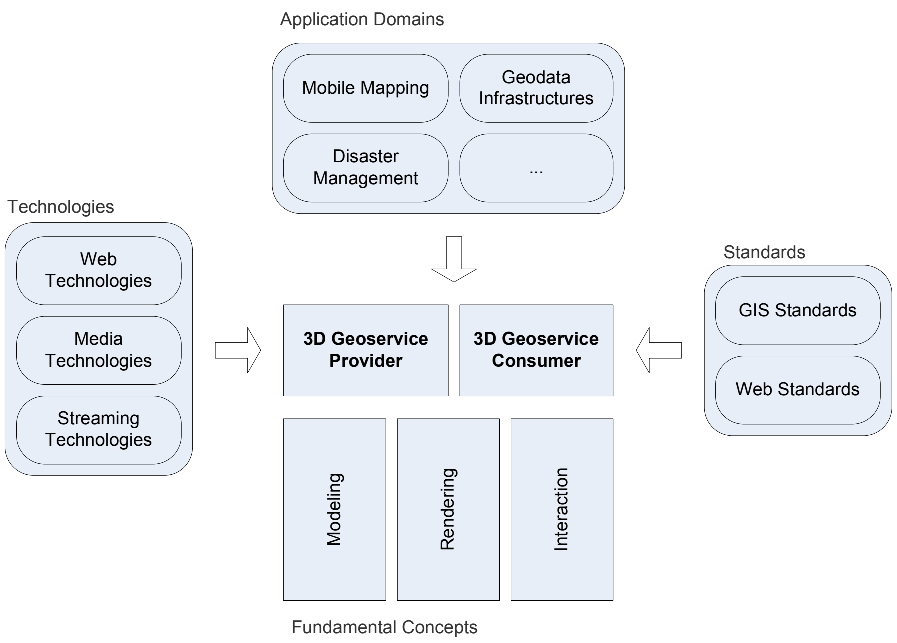


Figure 8: Context of geoinformation visualization in service-based systems.

As already described, this usage of geoservices is driven by a lot of application domains, as web mapping, disaster management, or the complex field of geodata infrastructures. According to the scenarios of these application domains, there must be appropriate services, which include the functional capabilities but also non-functional properties as availability, timeliness of the provided information or security issues. Especially for ad-hoc scenarios as they occur in the field of disaster management, it is important to have fast access to the most important information which means to avoid long-running information transfer but to provide easily consumable information units.

These computer graphics foundations are accompanied by additional technologies for making the rendering results consumable for the service consumer via web. Media technologies define the output and transport format of 3D geoinformation services. Hand in hand with those are streaming technologies which target on the efficiency of transporting geoinformation which could be images but also further processable raw geoinformation.

Additionally to these functionality oriented technologies, there must be further ones which allow the service provider to offer the service capability. These are web technologies as they are necessary for a service according to the simple service-oriented architecture. These are technologies for describing the service's capabilities, and make it accessible for service consumers. On a more concrete level, web technologies address protocols for message transport or security.

4 Interoperable 3D Geovisualization

4.1 Service-based construction of 3D geovirtual environments

4.1.1 Utilization of CityGML

"CityGML is a common information model for the representation of 3D urban objects. It defines the classes and relations for the most relevant topographic objects in cities and regional models with respect to their geometrical, topological, semantical and appearance properties. Included are generalization hierarchies between thematic classes, aggregations, relations between objects, and spatial properties." (CityGML web). CityGML provides an application schema for the Geography Markup Language (GML), which itself "is an XML encoding […] for the transport and storage of geographic information […] including both the spatial and non-spatial properties of geographic features." (GML 2004)

CityGML considers geographic features as, buildings, inner and outer building parts (walls, doors, windows or roofs), vegetation, and city furniture. For supporting regional models CityGML supports a TIN-based relief. In addition the visualization aspects, CityGML enables applications to use the structural and semantical properties for simulations, analysis or spatial data mining.

Because of its open format, the XML representation, the broad consideration of urban objects, and the support of not only geometric attributes, CityGML seems to serve as a good platform for the service-based integration of different geoinformation: CityGML can be used as an exchange format for city models and landscape models but also can be used for the exchange of other urban information: Considering its geospatial position an arbitrary information entity can be transformed into a specific CityGML object as it is described by CityGML which can contain or encode the attribute values of the abstract information entity.

The usage of CityGML as a basis for the provision of geoinformation can lead to new insights into existing geoinformation and to new geoinformation-based applications. Existing city- and landscape models can be used in new ways. As an example, CityGML models contain several aspects of a building information model (BIM). Its visualization might allow the end-user to have different views on the model for retrieving information about geometry, topology, paths, or different attributes. An other example is the utilization of the city models semantics for the development of new interaction techniques, e.g., for navigation in the GeoVE as it is described below (see 4.2.1).

By the help of CityGML and an appropriate transformation capability, complex geospatial information can be combined and transformed into a visual form that can be easily perceived and analyzed by human.

As CityGML is a GML application schema and deals with geographic features it can be retrieved from WFS. At all, geoservices dealing with CityGML could provide one or more of the following CityGML-related functionality:

- Retrieving CityGML
- Creating CityGML (e.g., from a database)

- Modifying CityGML
- Transforming CityGML (into another format)

For proving the potential of CityGML as an integration platform the following issues should be investigated:

- Usability of CityGML for the provision via WFS, especially the focused access to parts of the model
- Special view on performance of CityGML on-the fly creation, WFS access, and CityGML transport and processing
- Impact of relatively large data, possibilities of compression, web service access patterns
- Usability of CityGML in the context of building information model analyzes
- Possibility of incremental access to the CityGML model for improving the usability to the end-user

4.1.2 Integration of other geoinformation sources

In addition to the CityGML support for regional urban environments, additional information sources must be considered as WCS (e.g., elevation data, epidemiological data, environmental information, etc.), WMS (e.g., aerial photos, satellite photos), WFS which provide geographic features as GML, or other geographic information services.

A geovirtual environment might have to be constructed of several of these information sources. Thereby the different interoperability levels named in section 3.2.1 and additional properties of the GeoVE have to be regarded. This means extra effort for checking the information's meta data, performing necessary transformations and for setting up the GeoVE (e.g., scaling, coordinate transformation, adjustment of terrain size, correction of view parameters as near plane or far plane).

According to the OGC reference model (Percivall 2003) the single information sources are integrated by an application service, which might be an application server or a fat client, or by other geoinformation services which add value to the base information and provide this via service interfaces to other consumers.

4.2 3D Client Development

Geoinformation services provide distributed computing capabilities for being used in applications which do not posses the necessary data or processing capabilities. E.g., a 3D geoinformation service response may include an image of a 3D scene.

But accessing processing services or portrayal services needs time for finding or calculating the response, possibly for transforming it into an interoperable format, and especially for transmitting it to the service consumer.

If an end-user is involved into the system, this might be critical for the usability of the whole service-based system: When the user has to wait too long for an answer, he or she gets frustrated and might even decline the application. Thus we assume two things to be very important for the design and implementation of geovisualization services. These are on the one hand performance improvements for increasing the perceived speed of the application and on the other hand interaction improvements to handle the restrictions, to minimize them and to increase the overall usability of the geoservice-based system.

4.2.1 Interaction with 3D Geoservices

Interaction in 3D geovirtual environments targets on the analysis, editing, and navigation of the presented information space. The user interaction in geoservice-based GeoVE is not yet deeply investigated. We assume that we can find appropriate interaction techniques for such systems. Herby we will take into account the type of the presented information and try to utilize semantic information which is provided by CityGML models. We will also take into account the external properties of the service-based systems, e.g., the platform on which the services are running.

Possible interaction techniques that address these issues are a guided navigation which provides landmark based animation tours, semantic maps which allow the user to navigate step-by-step through the environment or the sketch-based navigation as it was introduced by Döllner et al. (2005). This sketch-based navigation is shortly described in the following paragraph.

Navigation as an example 3D geoservices interaction
As a key interaction, efficient navigation techniques are a crucial requirement in using geovirtual environments. Only navigation allows the user to explore and perceive the presented information. Darken and Siebert (1996) suggest three types of navigation tasks. These are naïve search, targeted search or exploration.

As one issue of interaction, navigation has to be considered in the context of portrayal services. Navigation integration is imaginable as an additional operation in portrayal services or as an extra service that attends to other service invocations.

As an example Döllner et al. (2005) have integrated the so-called sketch-based navigation technique into a movie-based portrayal service. This navigation technique uses pen-based strokes and gestures as input. These strokes and gestures are reprojected into the presented 3D scene and the intersections with the geovirtual environment are determined. Basing on the intersections and the navigation affordances of the hit geoobjects, an appropriate navigation handler is used which determines a camera path according to the input sketch and generates a corresponding navigation animation which is sent to the user immediately. This usage of the inherent navigation affordances of objects of the GeoVE has been further advanced by Döllner and Hagedorn (2006) the integration of visual navigation cues that give the user an experience of the pending navigation, e.g. arrows indicating the camera path and special symbols indicating directions where to look at – see Figure 9. Especially for the usability of 3D city models on mobile devices, this sketch-based navigation and the visual cues can provide an added value to the user.

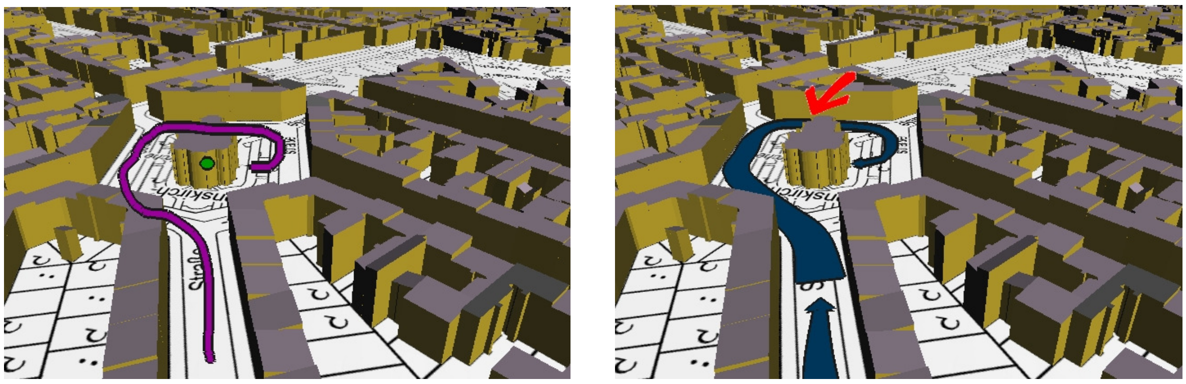


Figure 9: Example of a sketch-based navigation command (left) and the resulting visual cues integrated in the 3D scene. (Data: Official city model of the city Berlin, Germany.)

4.2.2 Performance improvements

3D geoservices provide integrated geoinformation as visualization for a variety of platforms, e.g. mobile devices. But for giving the user the feeling of control over the application, visualizations must be presented in a specific time and animations must be running with a specific frame rate. If model data are transmitted they should have small amount.
Under the conditions of limited bandwidth, we will investigate possibilities for improving the performance of 3D geoservice. This might include research in the following fields:

- Thinning of the 3D model: Elements of the city model that are obviously not interesting for the user are removed from the model for reducing the response payload.
- Generalization of the 3D model: Several single buildings might be summarized to a more general representation with less geometry which leads again to a reduced response payload. Currently generalization is a time consuming operation, so there must be a trade-off between effort and final speed improvements. As one approach generalizations could be pre-computed.
- For the provision of animations we will investigate different video encodings and the corresponding encoding and decoding processes.
- Data compression might be further useful for reducing the response payload. This can be done on the application layer with specialized compression methods or corresponding to the transfer protocol on transfer layer, too.
- Different transport mechanisms can be investigated in consideration of their throughput.

5 Integrative 3D Viewer Client

The OGC Web Services Initiative, Phase 4, (OWS-4) is an interoperability program set up by the OGC. The target of this initiative is to evaluate and advantage the interoperability of different OGC standards concerning services and data formats but also the interoperability of different vendors and their client and server implementations. We are working for the CAD/GIS/BIM thread of the OWS-4. This chapter reports the current efforts in OWS-4 and gives an overview on the scenario and the viewer.

5.1 Project overview

The CAD/GIS/BIM thread, deals with the interoperability of building information across the building lifecycle and also between information models from different communities:

- *Computer Aided Design* (CAD): CAD data provide a very detailed, geometry-oriented view on a building site which can be used for the construction of new buildings.
- *Geographic Information Systems* (GIS): GIS support a more general, not so detailed view on geospatial data. GIS data models include geometry, topology, and thematic attributes.
- *Building Information Model* (BIM): A BIM is a very rich and detailed view on a building. A BIM includes geometry and domain-specific information. E.g., a BIM includes information about the spaces of a building which might be rooms, corridors, floors. One example for a BIM is the Industry Foundation Classes (IFC).

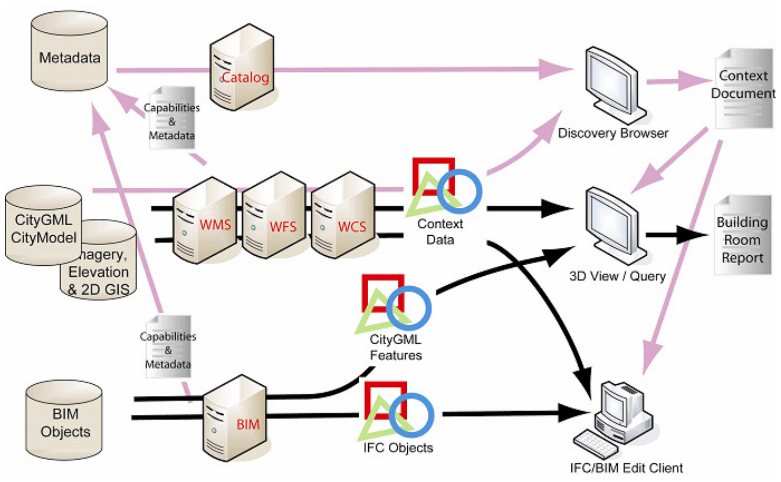


Figure 10: Generic solution architecture for thread CAD/GIS/BIM of OGC Web Services initiative, Phase 4. (Cote 2006.)

The overall scenario of the CAD/GIS/BIM thread is about the necessity of building up a field hospital in a military surrounding. From the scenario description (OWS4b) the generic architecture depicted in Figure 10 was derived. The scenario includes several actors and tasks: An analyst searches via discovery browser a metadata repository for retrieving first information about the occupied site. This information is recorded in a context document which later will be a basis for engineers to plan the hospital, a helipad, and surrounding sensors. Building models are provided by the so-called BIM server which will be realized as a special web feature service. This BIM server provides the building information model for editing in an appropriate BIM edit client which can store the edited model in the BIM server later. For getting an impression of the scene and enable further insight and assessment, the architecture provides a 3D view client which is capable of retrieving different geoinformation from different OGC web services (WMS, WFS) but also to access building information from the BIM server. This 3D view client is the contribution of our working group to OWS-4. With CityGML an additional building information exchange format is integrated and tested for interoperability.

5.2 3D View Client

5.2.1 Technical Basis

The 3D view client is implemented as a plug-in to the LandXplorer CityGML Viewer that itself is based upon the LandXplorer framework which is a real-time 3D geovisualization system. The LandXplorer system allows the creation, management, and visualization of large-scale 3D geovirtual environments. The system uses state-of-the-art real-time 3D computer graphics algorithms and offers efficient interactions with the geovirtual environment. This includes enabling/disabling of information entities, blending techniques for raster layers, access to object attributes, or a variety of navigation techniques. The CityGML Viewer is a software system that is capable of reading, loading, and writing CityGML data as they are described in the latest OGC discussion paper on CityGML (Gröger 2006).

5.2.2 Requirements

In addition to the already existing functionality of the LandXplorer 3D geovisualization system the following functionalities are needed to be provided by the 3D CityGML Viewer Client in the context of OWS-4:

- *Import, display, and activation of context documents*: Context documents describe which servers and which information entities to request. So context documents are a part of the run-time service binding. As the OGC has specified only a web map context document and context descriptions for other service types (WFS or BIM server) are pending, we have to define an own format for describing the binding and the functionality to request from the service providers.
- *Accessing WMS and WFS*: According to the OGC implementation specifications for WMS and WFS, we integrated a client stub for each of the service types.
- *Accessing CityGML from BIM server*: The BIM server bases on a WFS but has additional operations that must be considered for the client stub implementation. For web feature services a detection of the syntactical format (GML, CityGML) should be integrated.
- *Geodata processing*: One essential task for the integration of geoinformation from different providers is the projection into a unique spatial reference system by coordinate transformation. Furthermore the appropriate target spatial reference system must be determined.

- *3D geoinformation visualization*: For allowing a human user to get insight into the loaded data they must be visualized. As described this includes the mapping to a computer graphical representation which can be used by the 3D rendering system to synthesize a visualization of the integrated 3D scene.
- *Building room report*: In the described scenario, the building room report is necessary for the assessment of a building and it's capabilities for housing a field hospital.

5.3 Current Results

According to the proposal for participation in OWS-4 a demonstration implementation of the extended LandXplorer 3D CityGML Viewer Client has been provided to the OGC. The effort so far has been concentrated on the web context document processing, accessing WMS for retrieving map layers, accessing WFS for retrieving building model data in the CityGML format, and integration and visualizing these information.

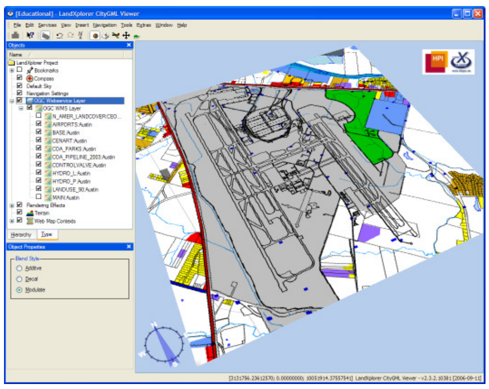
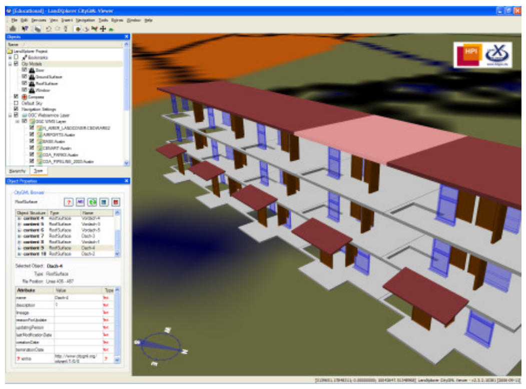
a) 
b) 

Figure 11: a) Combination of map layers from different WMS. b) Combination and integration of map layer information and feature data.

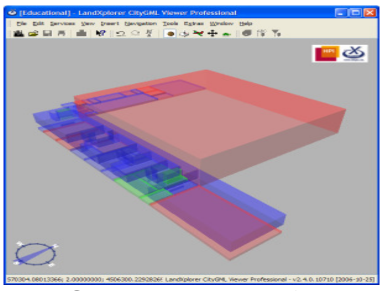


Figure 12: Visual room report. Color coding of building rooms according to the security state.

For describing which features to load and from which service they are provided, an arbitrary text format is used. It is leaned against the already existing Web Map Context specification. It is possible to load and execute a specific context description and load maps and CityGML features.

For the integration of geoinformation that are provided by different services, coordinate transformation is essential. This requires choosing a spatial reference system that can be applied to all data sets.

For some service requests the transfer volume is a very large one and so the synchronous service invocations are very time intensive. This will be a starting point for investigating techniques for optimizing the communication in the geoservice-based system.

Figure 11 shows an example view that is assembled from several single web map layer calls (a) and illustrates the combination of map layer information with feature data (b). In this special case the spatial reference systems had to be harmonized. Figure 12 shows the prototype of a visual room report. It enables to color the rooms of a building according to the value of a selectable attribute.

6 Conclusion and Future Work

The work so far has been about getting in touch with service-oriented computing, especially in the field of geoinformation. This included to get an overview about existing geoservices and their relationship to the common service-oriented architecture and to web services as a special and widely distributed form of implementing a SOA. It can be stated that service composition is important for achieving higher-level services with a higher value to the human user or another geoinformation processing system.

In the field of geoservices the OGC is a very active organization that targets on the interoperability of geoservices and geodata. The OGC has specified different services that cover different parts of the geovisualization pipeline.

The further work will bother with the processing of 3D geoinformation in geoservice-based systems. A focus will be on visualization services. As a new approach the interaction with visualizations, e.g., 3D GeoVE, will be investigated for integrating as a service. One issue in the assembly of higher-level functionality is the composition of services. This will be done in the field of 3D geoinformation services. As an outcome there will be concrete view clients using geoservices, concrete services processing 3D geoinformation, concrete service compositions of 3D geoservices, and derived and proved conclusions about the design and patterns for achieving this.

References

[1] Nadine Alameh. Chaining Geogrphic Information Web Services. In *Internet Computing, IEEE*, Sept.-Oct. 2003, pp- 22-29. IEEE Computer Society, 2003.

[2] A. Altmaier and Th. Kolbe: Applications and Solutions for Interoperable 3d Geo-Visualization. In: Fritsch, Dieter (ed.): Proceedings of the Photogrammetric Week 2003 in Stuttgart, Wichmann Verlag, Stuttgart, 2003.

[3] G. Andrienko and N. Andrienko. Knowledge-based visualization to support spatial data mining. In *Proceedings of the 3rd Symposium on Intelligent Data Analysis*, pages 149--160, Amsterdam, The Netherlands, August 1999.

[4] N. Andrienko and G. Andrienko. A framework for decision-centred visualisation in civil crisis management. In *Location based services & telecartography, Proceedings of the symposium* 2005, Vienna, 2005.

[5] C. Barham, et al. (eds.). ebXML Glossary. UN/CEFACT and OASIS, Version 0.99, 2001.

[6] Y. Bishr. Overcoming the semantic and other barriers to GIS interoperability. In Int. Journal on Geogr. Information Science, Vol. 12, No. 4, 1998.

[7] K. W. Brodlie, L.A. Carpenter, R.A. Earnshaw, J.R. Gallop, R.J. Hubbard, A.M. Mumford, C.D. Osland, and P. Quarendon (eds). Scientific Visualization, Techniques and Applications, 1992, Springer-Verlag.

[8] CityGML Homepage. Internet www.citygml.org (25.11.2007).

[9] Paul B. Cote. OWS-4 CGB Component & Architecture Notes, Version 0.2, July 2006, unpublished.

[10] S. Cox et al. (eds.). OpenGIS – Geography Markup Language (GML), Implementation Specification. Version 3.1.0, OpenGIS® Recommendation Paper, February 2004. Internet: http://www.opengeospatial.org/standards/gml (24.11.2007).

[11] R.P. Darken and J.L. Siebert. Navigating Large Virtual Spaces. In *International Journal of Human-Computer Interaction*, Januar-März 1996, 8(1), S. 49-72.

[12] Jürgen Döllner and Benjamin Hagedorn. Sketch-Based Navigation in Virtual 3D Environments. 2006. unpublished.

[13] J. Döllner, B. Hagedorn, and St. Schmidt. An Approach towards Semantics-Based Navigation in 3D City Models on Mobile Devices. In *Proceedings of the 3rd Symposium on LBS & TeleCartography*, Vienna, Nov. 2005, pp. 171-176.

[14] J. Fitzke, Claus Rinner, and Dirk Schmidt. GIS-Anwendungen im Internet. In *Geo-Informations-Systeme* 6/97, pp. 25-3, 1997.

[15] Gerhard Gröger and Thomas H. Kolbe. Interoperabilität in einer 3D-Geodateninfrastruktur. Münsteraner GI-Tage. 2003.

[16] Gerhard Gröger, Thomas H. Kolbe, and Angela Czerwinski (eds.). Candidate OpenGIS CityGML Implementation Specification (City Geographic Markup Language), Version 0.3.0, August 2006. Internet: http://www.opengeospatial.org/standards/dp (25.11.2006)

[17] ISO 19119, Geographic information Service. International Organisation for Standardization. 2005.

[18] OWS4b: No author. Request for quotation and call for participation in the OGC Web Services 4 Initiative, Initial operating capability and demonstration, Annex B: OWS-4 Architecture. 2006. Internet: http://www.opengeospatial.org/projects/initiatives/ows-4 (25.11.2006)

[19] G. Percivall (ed.). OGC Reference Model. Version 0.1.3, Open Geospatial Consortium. September 2003.

[20] B. Schmidt, M. May, and C. Uhlenküken. Dienste-basierte Architekturen für die Web-basierte 3D-Geovisualisierung. Münsteraner GI-Tage. 2003.

[21] R. Spence. Information Visualization, Addison Wesley, 2001.

[22] Workflow Management Coalition. Terminology & Glossary. The Workflow Management Coalition Specification, Issue 3.0, February 1999. Internet: http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf

Operating System Abstractions for Service-Based Systems

Michael Schöbel
michael.schoebel@hpi.uni-potsdam.de

From an operating system point of view, service-based computing can be seen as the execution of cooperating tasks or processes. Special properties or (meta-) information concerning the structure of such service-based systems are ignored on the operating system level most of the time.

More appropriate operating system abstractions can lead to optimized server implementations by exploiting special properties and knowledge about the structure of service-based systems. Optimizations are possible in the areas of scheduling, memory management or virtualization concepts.

Envisioned research results will lead to an abstract model of service-based systems from an operating system point of view. Tools and operating system extensions, which prove the applicability of the findings will be developed using the "Windows Research Kernel" (WRK) as well as the Linux operating system.

1 Introduction

Service-oriented computing is the computing paradigm that utilizes services as fundamental elements for developing applications [11]. Services are offered by a supplier (= service provider) and can be used by clients (= service consumer) to get the requested functionality.

Currently the IT infrastructure in many companies is restructured in a service-oriented way. Business functionality for internal use (e.g. human resource management) or external use (e.g. order acceptance) is provided as service which can be consumed in a standardized way. By combining services in different structures, flexible and easily adaptable IT processes can be realized, even across enterprise boundaries.

From a technical point of view service-orientation is nothing new. Concepts like modules, components, remote procedure calls and client/server computing are old and well known. But the new hype of service-orientation leads to another advantage: today IT infrastructure concepts can be described in terms business people understand. By bringing business and IT people together, optimized computing structures can be developed.

Other recent trends in the computer industry are platform independency and consolidation/virtualization. Platform independent applications and technologies can be easily migrated to infrastructures from different vendors or to machines with different operating systems. Consolidation/virtualization aims at better utilization of available computing power.

Operating systems as basic foundation for applications and servers should consider these recent trends. Most operating system abstractions like process, thread or virtual memory were not designed explicitly for (virtual) server systems or dynamic service-based environments (e.g. [2] or [9]). To what extent these concepts and abstractions can be improved considering the special properties of service-based systems is a promising research area.

This paper describes some ideas for new or better operating systems abstractions to support service-based systems. The central question is: How would the perfect operating system for service-based systems look like if it were designed from scratch? The intention is not to develop a new operating system, but to investigate new approaches and to integrate them into existing systems to improve performance, managebility or other properties of service-based systems.

The remainder of this section clarifies the term *service-based system* as we understand and use it from an operating system point of view. In section 2 current server implementation techniques regarding request processing and resource virtualization are described. Afterwards, in 3, a new operating system abstraction *batch scheduling* is introduced. Section 4 concludes the paper with an outlook on future work.

1.1 Service-based systems

In general, a service is something a client can use to get required functionality. The client needs to know how to find and how to access a service provider offering the needed functions. It is irrelevant for the client in which way the service provider actually executes the request (e.g. if another service is used), furthermore it is irrelevant for the provider what the client uses the results for. Therefore a loosely coupled structure of service providers and clients can be created which is called *service-based system*.

Recently, in the context of IT systems, the term "service" is often considered synonymous with "web-service". But service is a more general concept, as described above. Web-services are a specific implementation possibility for a service-based system with specific protocols like WSDL, UDDI, HTTP, and SOAP. We consider the term *service* in its general meaning. Hence a printer device driver with an appropriate interface can be considered as a service and can be part of a service-based system.

Short: service-based systems contain *services*. A service in this sense is a software entity, that can be accessed through standardized (and therefore machine processable) interfaces. Figure 1 depicts the conceptual view on services.

Services provide ports where clients can invoke functionalities. The service processes the clients' request and returns the results. The service implementation may require external services which are called during request processing. In such cases, the service itself acts as a client and sends a request to another service.

If a service does not require external services it is called *self-contained*. Client requests can be processed completely inside such a service.

A service can be either *stateful* or *stateless*. The service state refers to possible relations of subsequent client requests (= a session). To use stateful services the client has to establish an explicit connection for session initialization. Within this session

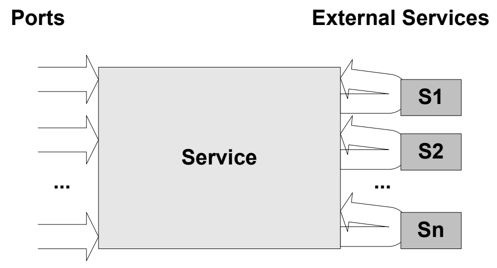


Figure 1: Conceptual service view

implicit information which the client do not have to send with every request can exist. Stateless services expect requests to their input ports which are complete and contain all necessary information for processing the request.

Most service-based systems utilize an application server to manage and host services. Application servers simplify generic service management tasks, e.g. service configuration, deployment, connection (with other services) or monitoring. They are often called *service containers*.

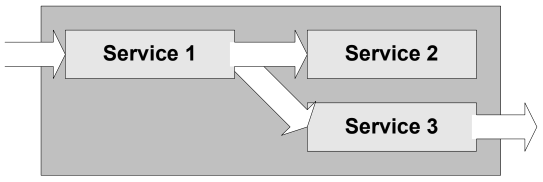


Figure 2: Service container - direct composition

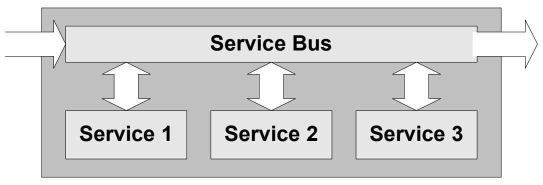


Figure 3: Service container - composition via service bus

Figure 2 and figure 3 show two basic structures of service containers which contain multiple services. The first possibility is known from the component programming paradigm: services are explicitly connected with other required services at deployment time; of course matching services must be available. The second possibility is the service bus concept. All deployed services are connected to a central bus which is responsible for the routing of requests to an appropriate service. With a bus structure more complex interaction patterns (e.g. publish/subscribe or content-based routing of requests) are possible.

The operating system can act as a service container for uncommon services like device drivers. It provides the required functionality to manage the drivers/services.

1.2 Special properties of service-based systems

With current abstractions from the operating system point of view, a service-based system is a set of processes with assigned resources (e.g. memory or CPU time).

Compared to common applications, services or service-based systems build up a different application model: A (less complex) client program is executed which depends on distributed/remote services. The effectively used services could be dynamically chosen at runtime.

This application model features special properties which must be considered when executing such an application. The availability of remote services for example is important.

(Meta-)information concerning the implementation of services (e.g. programming language, expected CPU usage or memory consumption) is available in the majority of cases. This can be important information for an optimized service execution for the server machine which hosts a specific service instance.

An operating system optimized for service-based systems should consider the special properties of such systems. Following, some aspects are described which can be exploited for optimizations.

- A *service composition* contains several services which are used together, e.g. one service calls another service to fulfill its task. Dependent services can be visualized as service composition graph.

  To support service compositions on operating system level, service composition graphs can be mapped onto operating system notions, such as process groups, jobs, or task sets.

- *Service level agreements (SLAs)* are used to describe a contract between a service consumer and a service provider. Both agree on a certain level of service delivery in terms of availability, pricing, or timing constraints.

  Considering SLAs on operating system level can help to optimize scheduling, e.g. execute specific processes for a specific client to reach the SLA or drop requests with a low priority.

In the following section existing operating system concepts are described taking service-based systems into consideration.

2 Related Work

This section describes related work in the proposed research area. First, request processing and scheduling in current server systems are described. Afterwards, a survey of different virtualization concepts is given.

The central runtime abstractions of current operating systems are the process and the thread concept. In general, a service in a service-based system is provided by multiple processes or threads (application server, service implementation, other used services, databases etc.).

Server developers think about server implementations in terms of request processing, availability, or reliability. Much effort is invested into realizing these requirements with means provided by the operating system.

An adjustment of operating system functionality to service-based systems can ease the development of such systems. By analyzing the current implementations of server systems unfavorable operating system abstractions/APIs can be identified.

2.1 Request processing in current server systems

A server can be seen as a stream processor: it receives a stream of requests, processes them and sends back a stream of results. In current server implementations different strategies of how the request processing is done exists.

2.1.1 Threaded request processing

Servers using the pure threaded request processing approach create a thread for each client request. The whole request and maybe its subsequent ones from the same client are processed in the context of this newly created thread.

Advantages of this approach are a good parallelization of request processing and a good usage of available resources. The high overhead of the thread lifecycle (creation, management, deletion) is disadvantageous.

To alleviate these drawbacks, concepts like thread pools are used and a specific number of request processing threads are created in advance and are dynamically assigned to incoming threads. In this case the server has to decide what to do with new requests if all worker threads are busy.

2.1.2 Event-driven request processing

Pure event-driven servers process all requests within a single worker thread. Arriving requests are stored in a queue which is used as input for the worker thread. The worker fetches and processes the requests one at a time.

That no per request overhead is introduced and that the queuing policy can be defined to ensure e.g. request priorities is advantageous. As a disadvantage the low degree of parallelism must be mentioned: the CPU is idle if the request processing requires blocking system operations.

2.1.3 Design patterns for request processing

Pure threaded or event-driven server implementations are rarely used. Instead, different design patterns lead to hybrid server implementations with aspects from both types.

Staged request processing A possible way to think about request processing is to imagine that every request has to pass several *stages* of processing until the response can be sent back to the client. This idea is independent from a threaded or event-driven approach and can be used in both models.

Depending on the specific server the stages can be more or less clearly identified. A thread in a threaded server could call multiple functions which represent different processing stages. In an event-driven system, a new event indicating the next processing stage (e.e. connection establishment, request, response) can be used to distinguish different stages.

In [8] and [6] special server optimizations are described which exploit the stages to efficiently schedule the processing threads. Additionally, the used staged server pattern are described.

A special scheduling approach called "cohort scheduling" for staged server applications is described in section 2.2.1.

Thread pools The idea of thread pools is that a specific maximum number of threads is used for request processing. These threads can be initialized at server startup and are dynamically assigned to incoming requests. If all threads are busy and a new request arrives the server could store the request in a queue or drop it. In this way the per request overhead of thread creation has vanished. Furthermore, a better estimation of required (operating system) resources is possible, because the number of threads is known in advance.

Reactor pattern A simple implementation of event-driven request processing involves a single worker thread. This thread fetches incoming requests, checks the request type and calls an appropriate processing function.

The reactor pattern (see [15]) separates the event dispatching from the application-specific request processing. A *reactor* is waiting for different events from multiple event sources (e.g. network sockets). The (synchronous) waiting can be implemented using operating system API functions such as `select()` or `WaitForMultipleObjects()`. If one event source indicates a pending event, the reactor calls the registered event handler for this event.

In a HTTP server implementation a reactor can wait for incoming connection requests on a specific socket. If such a request arrives, a handler for the new connection is created. This newly created handler is responsible for client requests from the specific source address.

The main advantage of using the reactor design pattern is the separation of application independent demultiplexing and dispatching mechanisms from application specific event handling functionality. Additional event handlers or different dispatching strategies can be easily integrated. Furthermore, the reactor pattern provides a coarse grained concurrency control on event dispatching level. A single thread of control calls the event handler and serializes the event processing in this way.

The single threaded request processing is a big drawback if the event handler does many blocking I/O operations. While waiting for I/O completion the CPU is idle and no other request is processed. Another slight restriction of the reactor pattern is that it can only deal with event sources for which the operating system provides synchronous waiting functions such as `select()`.

If the reactor calls the event handler functions asynchronously or dispatches the incoming requests to multiple threads, a combination of event-driven and threaded request processing can be achieved and some disadvantages can be alleviated.

Proactor pattern If the server operating system provides asynchronous I/O API functions, the *proactor* pattern (see [15]) can be used to realize a concurrent request processing with asynchronous functions.

An application creates a handler for asynchronous function calls. Such a handler is registered at the proactor. If an asynchronous function call gets completed, the proactor dispatches the completion event to the registered handler. In this way many concurrent I/O functions (and therefore client requests) can be processed concurrently by using operating system capabilities.

A HTTP server implementation could provide several handlers for different processing events, such as connection requests or file transfer. A single handler dispatches an asynchronous operating system call and dispatches the next request. A corresponding completion handler is called if the operating system finishes the asynchronous call.

The proactor pattern decouples threading and concurrency. By using asynchronous function calls and I/O completion callbacks a concurrent request processing is possible without using threads explicitly. In fact, there is still only one thread of execution. Therefore, the general advantages of event-driven request processing are still valid: simple application synchronization and no thread management overhead.

To apply the proactor pattern successfully, native operating system support for asynchronous I/O processing is required. The pattern relies on the operating system implementation of asynchronous functions, therefore some configuration possibilities for request processing (e.g. prioritize or abort pending requests) can be hard to achieve compared to a threaded approach.

Event-driven servers can process multiple requests concurrently by using the proactor pattern. In this way, a main drawback of event-driven servers with a single worker thread is resolved.

2.1.4 Summary

The main concepts are the threaded and the event-driven request processing concept. A performance comparison of these two approaches for Java servers can be found in [4]. Both approaches have several drawbacks, therefore the authors argue that hybrid servers with features from both approaches are necessary for higher performance.

In [20] the authors propose that threaded and event-driven implementation approaches are at the opposite ends of a design spectrum and the best implementation strategy is somewhere in between. Event-driven server implementation techniques are useful for high concurrency and lock-free request processing. For dealing with multiprocessor parallelism and blocking I/O, threads are needed.

With *staged event-driven architecture* (SEDA) [19], a method for server implementation is provided which combines event-driven and threaded request processing.

A server application consists of a set of stages. Each stage has an assigned thread pool and an event queue. Controllers are used to adapt the stage behaviour in special situations, especially at overload conditions. A thread pool controller observes the length of the event queue and creates new threads if necessary. A batching controller determines the number of processed events per event handler loop by observing the throughput rate.

In [18] the authors explain why current operating systems do not provide much support for high-performance internet servers: First, thread-based concurrency models lead to high overhead in terms of context switch time and memory footprint. Furthermore, thread-based scheduling allows only a coarse grained resource usage control. Second, blocking I/O operations limit the number of concurrent activities. And third, operating systems hide their performance related optimizations (e.g. file system buffer cache) inside the kernel and do not allow an application specific optimization policy.

In short, the authors believe that the virtualization of hardware resources provided by operating systems is not appropriate for highly concurrent server applications. Such applications require a fine grained, low-level access to hardware resources like the CPU or the main memory. The operating system should provide low level access functions which allows the implementation of application specific optimizations.

In [17] a detailed argumentation why the event and thread paradigms are equivalent is given. The authors argue that the drawbacks of threading compared to events are artifacts of specific threading implementations and not inherent to the threading paradigm. They conclude that threads are the better concept: event-driven approaches can be mapped on threads, threads are the better abstraction for developing concurrent systems (e.g. better debugging possible) and compiler support can lead to more optimization of the applications.

Their conclusions are based on two assumptions about modern server systems: First, the concurrency in modern servers results from concurrent requests that are largely independent. Second, the code that handles each request is usually sequential. For such systems the thread abstraction is more natural. Further they propose the usage of highly scalable user mode threading models to prevent the large thread management overhead introduced by operating system threads. User mode threads can be optimized by compiler tools which are able to detect faults and optimize locking and similar synchronization problems.

2.2 Task scheduling in current server systems

The scheduling of concurrent activities is closely connected to the general concept of request processing in server systems. The server developer has only slight influence on the operating system scheduling policy if no user mode thread library is used. Therefore, some different scheduling approaches were proposed to especially support server systems.

2.2.1 Cohort scheduling

Besides the staged computation in server systems (see section 2.1.3) a matching scheduling policy called *cohort scheduling* is described in [8].

If a server processes incoming requests by channeling them through different stages, this fact can be exploited to maximize the benefit from modern CPU concepts like caches, TLBs and branch predictors. Caches and different prediction heuristics lose a lot of their effectiveness if the CPU makes a context switch to a different, completly unrelated thread. The idea of cohort scheduling is to process multiple threads which are in the same execution state (= in the same processing stage) sequentially and to keep caches and branch prediction valid.

The authors describe a user level thread library which implements the proposed concepts and report an achievable server throughput improvement of 20% by reducing the processor cycles per instruction by 30% and L2 cache misses by 50%.

2.2.2 Affinity Scheduling

An extension of the cohort scheduling approach is provided in [6]. The authors describe four different algorithms for cohort scheduling. The algorithms differ in the way requests are processed inside one single stage and in the way the point in time for the next stage is determined.

- Dynamic-gated: Every computation stage includes a request queue. If a stage is scheduled for execution, all requests in the input queue are put together to a batch of requests. Now this batch passes all computation stages sequentially.
- Threshold-gated: Again, every computation stage has a queue for incoming requests. The switch to the next stage happens when a specific number of requests are processed in the current stage or the queue is empty. Compared to the dynamic-gated policy an upper bound for the request number per batch is defined.
- Non-gated: This policy uses dynamic- or threshold-gated policy from the second computation stage. At the first stage the behaviour is different: All requests are processed until the request queue of the first stage is empty.
- Cutoff-gated: If a single request requires much time in one stage, all other requests are delayed. This policy defines a cutoff value for the useable CPU time of a single request in a single stage. If a request spends this amount of time in a computation stage its computation is preempted and the remaining request is put back into the request queue.

2.2.3 Coscheduling/Gangscheduling

Coscheduling or gangscheduling can be used for multithreaded applications on multiprocessor systems/clusters especially. The aim of this scheduling approach is to minimize the inter-process communication costs and synchronization costs.

A scheduler using gangscheduling assigns the different processes of an application to different processors simultaneously. For a specific time (quantum) the processes are parallely executed. After the quantum expires the next application is chosen and assigned to the available processors, the context switch is coordinated across the processors.

When using coscheduling, the different processors are scheduled separately. By considering inter-process communication the schedulers try to execute communicating processes at the same time, thus reducing the blocking time of a single process.

Different algorithms for these basic ideas try to improve the overall behavior of this approach by assigning processes to the same processor every time or by scheduling different applications at the same time if enough processors are available.

2.2.4 Scheduler activations

To support user mode and application specific scheduling policies [1] proposes a concept called "scheduler activations". If a scheduling decision is necessary the operating system kernel invokes a user mode callback. This user mode function decides which thread is scheduled next, so it can realize an application-specific scheduling and prevent the known drawbacks of user mode threading models.

For an optimized scheduling support the kernel needs information about application details (e.g. number of concurrent activities), and the applications need information about kernel events (e.g. processor reallocations and I/O requests). The authors describe a user mode threading library which interacts with the kernel.

The modified operating system scheduler provides virtual processors for each application. These processors are under application control and can be assigned to the application threads.

Scheduler activations are a flexible concept to realize application specific scheduling policies. The approach loses its practicability in a wide range of applications when considering the frequent kernel/user mode interactions: both parts have to exchange all information which can (possibly) lead to scheduling decisions.

2.2.5 Complete scheduling model

In the context of real time systems and predictable computing in [5], a unified scheduling model for precise computation control is described. Every computation entity (including interrupt service routine, asynchronous procedure calls, etc.) is part of a scheduling group. The operating system scheduler calls scheduling decision functions (SDF) which are provided in each scheduling group to determine the next computation.

The proposed scheduling structure leads to a very flexible scheduling model: application specific scheduling is possible, furthermore a precise control of all scheduling aspects (e.g. interrupt handling) can be achieved.

2.2.6 Summary

Scheduling on the operating system level and request processing in server applications are closely connected: Harmonizing both aspects is necessary for achieving high throughput and a good performance. Another related aspect are the synchronization methods used to coordinate concurrent activities.

A scheduling and request processing approach explicitly designed for service-based systems does not exist. Section 3 describes a starting point for such an approach.

2.3 Virtualization

A good utilization of available computing power is an important issue in current IT infrastructures. Virtualization technology can help to manage IT resources and to easily react on changing requirements.

Virtualization can be defined as "... a technology that combines or divides computing resources to present one or many operating environments using methodologies like hardware and software partitioning or aggregation, partial or complete machine simulation, emulation, time-sharing, and many others." (from [10])

These different aspects of virtual machine implementations rely on the underlying operating system. Currently "whole system" virtual machines get better support from the hardware (e.g. *Vanderpool*-technology from Intel or *Pacifica* from AMD). Other types of virtual machines could get better support from the operating system.

Especially multiple layers of virtualization lead to complex computing structures which offer a base for optimization. The basic idea is to remove virtualization layers (if possible) and expose the underlying concepts directly.

2.3.1 Virtualization technologies

This section provides an overview and different classification approaches for existing virtualization technologies.

In [10] the authors identified five different virtualization abstraction levels, which differ significantly in terms of performance, flexibility, ease of use, resource consumption, scalability, and therefore in their usage scenario.

- Instruction set level: Virtualization on instruction set level is achieved by emulating the complete instruction set architecture of a specific platform. The instructions are interpreted by software and translated to the instruction set of the host platform. To use virtualization at this level, the instruction set translation must be possible. (Examples: Bochs [1], QEMU [2])

- Hardware abstraction layer level: A virtual machine monitor (VMM) is introduced between hardware and operating system level. The installed operating systems

[1]http://bochs.sourceforge.net/
[2]http://fabrice.bellard.free.fr/qemu/

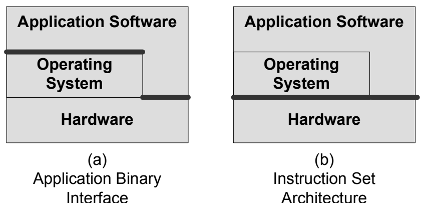


Figure 4: Virtualized system interface

use a virtualized view of the hardware which can be influenced by the VMM configuration. (Examples: VMWare [3], VirtualPC [4], Xen [3])

- Operating system level: The interface of an operating system kernel is defined by a set of system service calls. Virtualization on operating system level is achieved by providing different system service calls for different applications. Every application can have its own view on the operating system. (Examples: Solaris Zones [13], FreeBSD Jails [7])

- Library level: Mostly, applications do not call the operating system kernel directly, instead they use libraries with wrapper functions. Hence, a virtual operating system environment for a specific application can be built if the required libraries are provided. The new implemented libraries can use different system service calls (even on totally different operating systems). (Examples: WINE [5], LxRun [6])

- Application level: Another approach to virtualization is to create a virtual machine/environment as an application inside a given operating system environment. Such virtual machines can be arbitrarily complex and are not intended to be transparently inserted into an existing software stack. They can require the usage of a special programming language. (Examples: Java Virtual Machine [7], Microsoft .NET CLI [8])

A different classification of virtual machines architectures using different terms is provided in [16].

First, virtual machines can be differentiated by whether they are *process virtual machines* or *system virtual machines*. Process virtual machines support an individual process and are created/terminated together with the guest process. System virtual machines provide a complete system environment and can support multiple processes.

Second, the virtualized system interface is considered. The interface provided by a virtual machine can either be on the *Instruction Set Architecture* (ISA) or on the *Application Binary Interface* (ABI) level (see figure 4). ISA describes the platform dependent hardware instruction set, including user level and privileged system instructions. An

[3]http://www.vmware.com/
[4]http://www.microsoft.com/windows/virtualpc/
[5]http://www.winehq.com/
[6]http://www.ugcs.caltech.edu/ steven/lxrun/
[7]http://java.sun.com/docs/books/vmspec/
[8]http://msdn.microsoft.com/netframework/ecma/

ABI does not contain the privileged instructions, but a set of operating system calls to wrap them.

Given those two attributes one can classify virtual machines according to their type (process or system VM) and whether they provide the same or a different ISA of their hosting system.

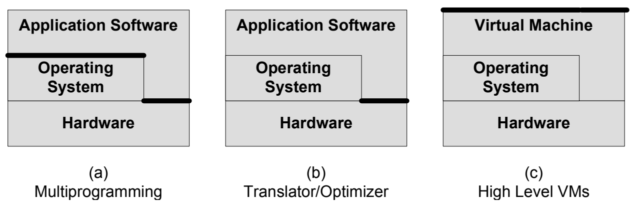


Figure 5: Process Virtual Machines

Process virtual machines can be subdivided into multiprogramming systems, emulation and dynamic binary translators, dynamic optimizer and high level virtual machines. The virtualized interfaces are illustrated in figure 5.

Multiprogramming environments allow multiple processes within one single operating system. The processes use the same operating system interface and the same type of instructions. Emulation or dynamic binary translation is necessary if a binary file compiled for a different platform is executed. The operating system interface is the same but the instructions have to be translated. The third type of process virtual machine use a given operating system environment and provides its own ISA. The Java virtual machine with its interpreted byte code is a good example for this type of VM.

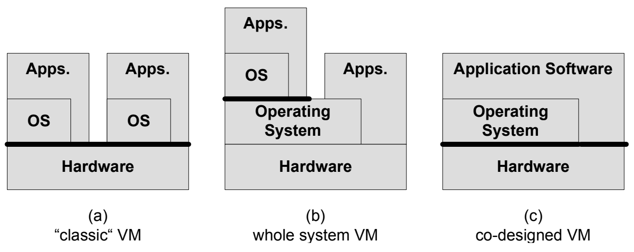


Figure 6: System Virtual Machines

System virtual machines can be subdivided into "classic" virtual machines, whole system VMs and co-designed VMs. The virtualized interfaces are illustrated in figure 6.

A "classic" virtual machine allows the parallel execution of multiple operating systems. The VM provides unique views of the hardware for every installation. Whole system VMs use the operating system services of a host system to build the hardware abstraction for the guest operating system. Co-designed VMs translate between different ISA systems; in this way an operating system can be executed on an ISA it was not designed for.

2.3.2 Virtualization in service-based systems

Currently there are many different types of applications of virtualization technologies ([10]).

- Server consolidation: Many server machines are under-utilized. Server consolidation aims at a better utilization of expensive systems. Furthermore the administration afford could be decreased.

  Virtualization supports server consolidation by providing the possibility to isolate different server applications. Depending on the server type, every installation have its own view of (a part of) the server machine.

- Secure computing platforms: Virtual machines can provide secure and isolated environments for applications and operating systems. A software failure or an attack only harms a virtual system which can be easily recovered.

- Kernel debugging and driver development: The target system of an operating system or a newly developed driver can be emulated by a virtual machine. In this way, software for an unavailable platform (e.g. a new CPU architecture or a new external device) can already be developed and tested. The ability of fast restarts makes development a lot easier.

Many other applications which are possible are not listed here.

In service-based systems virtualization can be found on many different levels: The trend towards server consolidation leads to more than one single operating system instance on a server machine. Furthermore, many service-based systems are using webservices in combination with a platform independent programming language (i.e. Java or .NET). The services are hosted inside an application server which can also be written in Java or .NET.

If we consider virtualization as a technology for dividing computing resources to present multiple operating environments (see [10]), most current server systems use virtualization. The complexity of such systems increases and the manageability or performance tuning is much more difficult than in non-virtualized environments.

2.3.3 Flattening virtualization layers

The general aim of virtualization (e.g. *virtual* memory management) is to provide an easy to use view of the available resources. Developers can use the simplified abstractions and implement complex applications without knowledge of certain aspects of the underlying hardware.

In [18] it is argued that virtualization in general is ill-suited for server systems. The authors consider the big advantage of virtualization as the biggest drawback: virtualization fundamentally hides the fact that resources are limited and shared. Their conclusion is that operating systems should eliminate the abstraction of transparent resource virtualization and should allow application specific and fine grained control over available resources.

Besides these basic considerations, the complexity of virtualized environments can be decreased by replacing heavy-weight virtualization with more light-weight approaches. Multiple isolated operating systems on a single machine, created by using a VMM like Xen [3] can be replaced by multiple views on a single operating system with concepts like Zones [13] or Jails [7].

3 New abstraction: batch scheduling

This section presents some thoughts on operating system improvements for a better support of service-based systems. The new operating system abstraction *batch scheduling* is introduced.

Current operating systems are designed for general purposes. Some server operating systems (like Windows 2003 Enterprise Edition) introduce slight optimizations for server applications. Examples are the modified scheduling quantum or the better support for multiprocessor platforms.

Typical service-based systems offer meta-information about their structure and execution. This information could be used to create an operating system especially suited for service-based systems.

The CPU is one of the most important resources determining server performance. Exploiting the special properties of service-based systems to optimize CPU utilization seems to be a very promising approach.

3.1 Problem description

A server receives a stream of client requests, does some computation and sends a stream of results back to the clients. Irrespective of the concrete server application, a high server throughput (i.e. process a maximum of client requests per time unit) is desirable.

Today, different components of service-based systems are optimized separately. The number of request processing threads is chosen or caches/other heuristics are used to optimize the server behaviour. Normally, a coordination across component boundaries does not exist.

Furthermore, inter component dependencies of different optimization steps are hard to predict and often ignored.

In general, a concept to combine different information about the service-based application (e.g. composition graphs or resource requirements) could lead to an optimized execution. An operating system designed for service-based systems should provide means for efficient task execution and coordination.

The following batch scheduling concept is a starting point for a new scheduling policy which offers several points for the integration of meta-information about the executed tasks/threads.

3.2 Solution approach

For a better execution control, the units of execution (in general threads or processes) which belong to a specific application or service should be collected. The operating system could provide means to manipulate and configure such collections as a whole. For example job objects or task sets could be used to define the behaviour of program execution.

Probably a more flexible way of execution control could be achieved if more fine grained constructs (e.g. threads instead of processes) are put together. Batch scheduling as proposed here shows a way to simultaneously manage and execute single threads.

In this section the proposed solution is described: First the general concept is introduced, afterwards the implementation idea is shown. Finally a possible realization of the implementation in the context of the Windows platform is described.

3.2.1 General concept

The idea is to assemble threads which are in a comparable state of execution together and schedule them in a special way. A comparable state of execution is given, when some threads access the same piece of data and/or execute the same sections of code.

A new operating system concept *batch* is introduced. A batch encapsulates similar activities across process and thread boundaries. It has a name and can be seen as a "global" operating system concept like "thread" or "mutex".

Regular operating system threads can be assigned to a specific batch. If, for example, a set of threads in a thread pool deals with client requests, the stages of request processing (parse request, load data from file, send result etc.) can be defined by the server developer and a batch can be created for each stage. All request handling threads which are in the same computation stage are put together into the specific batch.

The batch concept is similar to the thread concept. A batch can be initialized and can be executed by selecting one of the contained threads. The different thread states can be mapped to specific batch states. In this way batches can be treated as scheduling objects with a special semantic.

Threads which are assigned to a batch are scheduled indirectly: If the scheduler selects a batch for execution, the batch policy determines which operating system thread gets executed.

Basically, a batch policy has to describe three things: (1) Which conditions have to be fulfilled for the batch to be moved into the "ready for execution" state? (2) If the batch is chosen for execution by the scheduler, when does it leave the "run" state? (3) How is the next thread selected?

A point that has to be considered when selecting a specific policy for a batch is fairness and progress concerning other (not batched) threads in the system. If the batch never leaves the running state no other activities can ever be executed.

Following, some possible batch policies are described. Some of them are adapted from [6] and many other policies should be implementable.

- A batch collects threads and becomes ready for execution if the number of threads has reached a specific threshold N. After selection for execution the batch dispatches the first N threads in FIFO order. The threads run until completion or until transition to the wait state. Afterwards the batch yields the CPU and is re-inserted into the ready queue by the scheduler.
- To prevent a thread from blocking the CPU for an inopportune time the first policy can be modified: threads still run until completion or transition to the wait state, but they are preempted and reinserted into the batch after consuming a certain amount of CPU time.
- A batch can be always ready for execution, if at least one thread was inserted. If the scheduler chooses this batch, every thread in the queue is sequentially dispatched to the CPU and executed until completion or for a specific amount of time (batch quantum). The batch quantum can be different from the regular operating system quantum.

A server application can use batches to assemble request handling threads and to schedule them in a way more appropriate for CPUs with optimizing heuristics like branch prediction and data caches. Additionally the batch concept can be used to develop further operating system optimizations which can benefit from aggregated threads with similar behaviour.

3.2.2 Implementation concept

A thread batch is introduced as an operating system concept similar to threads. Like threads, batches have a priority and are scheduled for execution. The kernel manages batches and monitors their creation and management.

A new (kernel) datastructure is used to manage information about batch objects. It holds information about the batch execution policy and the batch priority. Furthermore a queue of threads is managed. Additionaly the thread datastructure is extended to store information about a batch object the specific thread is assigned to.

For using the batch concept, the operating system interface is extended with functions for batch creation and management.

- `CreateBatch` allocates and initializes (kernel) memory for a batch datastructure. The new batch gets a unique name and an initial priority and execution policy. The system call returns a handle to the batch for further operations.
- `Get/SetBatchPriority` modifies the batch priority. The priority is analog to thread priorities.
- `Get/SetBatchPolicy` modifies the batch execution policy. The application developer can choose from a set of predefined policies.

- `CloseBatch` removes the contained threads from the batch and frees the allocated memory for the batch datastructure.

After creating some batches and setting appropriate priorities and policies, server application threads can use the following system service calls to employ the special batch scheduling.

- `EnterBatch` puts the currently running thread into a specific batch queue. Afterwards, the thread releases the CPU and the scheduler chooses the next element for execution. The thread is now under batch control and runs if the batch is selected by the scheduler. If the running thread is already member of a batch (which is in this case currently running), it is put into the new batch and removed from the current batch.
- `LeaveBatchMode` explicitly dequeues the thread from the currently running batch. Afterwards the thread is again a "normal" operating system thread which is scheduled individually.

Both methods allow a flexible usage of the batch concept and a straight-forward implementation of staged request processing.
Inside the kernel three functions for batch manipulation have to be implemented.

- `KeInsertThreadBatch` gets a thread handle as parameter and puts the thread into the thread queue. This function is called by the `EnterBatch` implementation. Depending on the policy the batch can change its state and can become ready for execution.
- `KeRemoveThreadBatch` removes a specific thread from a batch. The thread batch field in the thread control block is set to null. Depending on the policy the batch can change its state after dequeueing a thread.
- `KeBatchGetNextThread` selects the next thread to run from the batch thread queue. The batch execution policy determines which thread to choose.

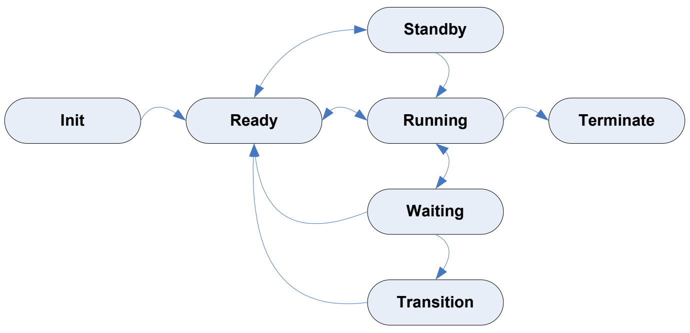


Figure 7: Windows thread states

During execution, threads change between different states. Figure 7 [14] depicts these states for thread execution on Windows systems.

The batch abstraction does not introduce new thread states. Instead, the batch concept introduces a second dimension to the thread states: Every thread can be in one of the displayed states but disclose a second information whether it is in a batch or not. The transition between the two levels is possible by using the described API functions `EnterBatch` and `LeaveBatchMode`.

Compared to the thread states batches itself can be in any of the states depicted in 7 except for the "transition" state. A thread is in the transition state if its kernel stack is paged out of memory. A batch does not have an explicit kernel stack, therefore the state is without meaning.

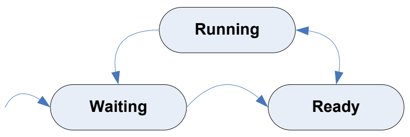


Figure 8: Simplified thread states model

For a detailed explanation of the proposed batch scheduling concept the simplified thread state model depicted in figure 8 is used. Without loss of generality threads can be in one of three states: (1) If the thread is currently executed it is in the "running" state. (2) If the thread is ready for execution but was not yet selected by the scheduler it is in the "ready" state. (3) If the thread is neither in execution nor ready for execution it is in the "waiting" state. This state integrates states like "standby", "transition", "waiting", "initialized" or "terminated", known from other thread state models.

Most current operating systems use a priority based, preemptive round robin scheduling algorithm. There are different priority levels, each one has its own ready queue. The thread with the highest priority in the ready state is selected for execution. The currently executed thread is preempted if a thread with a higher priority becomes ready. The thread in the running state executes for a specific time (quantum). Afterwards, it gets preempted and is placed at the end of the ready queue of its priority level. If a thread in the running state invokes a blocking system call (e.g. blocking I/O or semaphore operations) the thread goes into the wait state and the scheduler selects the next ready thread.

To integrate the batch scheduling concept into an existing round robin scheduling algorithm, the scheduling decision logic has to be adapted. Scheduling decisions are made as reaction to the following events:

- A thread finishes waiting and gets into the "ready" state. Depending on the priority of the new ready thread, the currently executing thread has to be preempted and replaced.
- The quantum of the currently running thread expires. The thread is placed into the ready queue and the next thread is dispatched for execution.
- The currently running thread changes into the "waiting" state. If a blocking system call is invoked or any other operation prevents the current thread from further execution a new thread has to be chosen.

First we examine the integration of batches into the general round robin concept.

Batches have a priority like every regular thread. They are inserted into the ready queues according to their priority. The round robin scheduler selects the thread from the ready queue with the highest priority. If a batch is selected it is marked as "in execution" and the `KeBatchGetNextThread` function is used to get the real thread to execute.

If the batch has finished its execution it is reinserted at the end of the ready queue of its priority. This conforms to the regular round robin algorithm. Depending on the batch processing policy of the executed batch, it can get into a state where it is not ready for further execution. In this case the batch enters the waiting state. Later it can become ready again and gets reinserted into a ready queue.

This procedure leads to an indirect scheduling of the threads which are part of a batch. Batches are building a meta-object, from the CPU point of view they are never executed. The regular scheduling algorithm has to be adapted for the case that a batch is selected for execution. Preemption and selection of the next thread has to be handled differently.

Now we can examine the special scheduling of threads which are part of a batch. As mentioned above three cases have to be considered.

- A thread finishes waiting and gets into the "ready" state.

  If the thread is not part of a batch it is handled by the regular round robin algorithm: the thread is inserted into the run queue of its priority level or preempts the current thread if its priority is higher. If currently a batch is selected for execution it can also be preempted.

  If the thread is part of a batch (regardless if it is currently in execution) it is reinserted into this batch by using `KeInsertThreadBatch`. Batch policy dependent, the specific batch becomes ready or performs some other action.

- The quantum of the currently running thread expires.

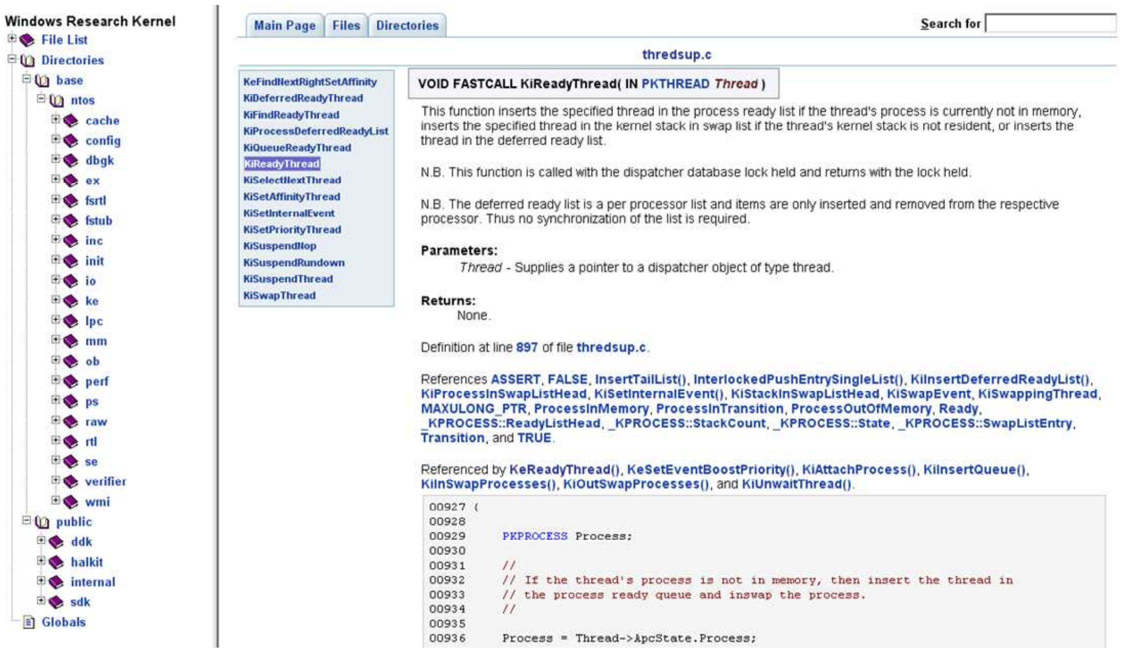
  The regular round robin algorithm is applied if the thread is not part of a batch: the scheduler puts the thread at the end of the ready queue and selects the next item to execute.

  If the thread is part of a batch the batch policy decides what has to be done.

- The currently running thread changes into the "waiting" state.

  If the running thread is not part of a batch and calls a blocking system service, the scheduler puts it into the appropriate wait queue and selects the next thread or batch for execution. A special case is the `EnterBatch` call: the calling thread is inserted into the specific batch before selecting the next item for execution.

  This special case has to be considered also if the running thread is part of a batch. Then the thread is removed from its old batch (`KeRemoveThreadBatch`) and inserted into the new batch. By calling `LeaveBatch` the thread also leaves the current batch. If a batched thread performs some other action which brings it to the waiting state, the scheduler puts the thread into a waiting queue and calls



thredsup.c

VOID FASTCALL KiReadyThread(IN PKTHREAD *Thread*)

This function inserts the specified thread in the process ready list if the thread's process is currently not in memory, inserts the specified thread in the kernel stack in swap list if the thread's kernel stack is not resident, or inserts the thread in the deferred ready list.

N.B. This function is called with the dispatcher database lock held and returns with the lock held.

N.B. The deferred ready list is a per processor list and items are only inserted and removed from the respective processor. Thus no synchronization of the list is required.

Parameters:
Thread - Supplies a pointer to a dispatcher object of type thread.

Returns:
None.

Definition at line 897 of file thredsup.c.

References ASSERT, FALSE, InsertTailList(), InterlockedPushEntrySingleList(), KiInsertDeferredReadyList(), KiProcessInSwapListHead, KiSetInternalEvent(), KiStackInSwapListHead, KiSwapEvent, KiSwappingThread, MAXULONG_PTR, ProcessInMemory, ProcessInTransition, ProcessOutOfMemory, Ready, _KPROCESS::ReadyListHead, _KPROCESS::StackCount, _KPROCESS::State, _KPROCESS::SwapListEntry, Transition, and TRUE.

Referenced by KeReadyThread(), KeSetEventBoostPriority(), KiAttachProcess(), KiInsertQueue(), KiInSwapProcesses(), KiOutSwapProcesses(), and KiUnwaitThread().

```
{

    PKPROCESS Process;

    //
    // If the thread's process is not in memory, then insert the thread in
    // the process ready queue and inswap the process.
    //

    Process = Thread->ApcState.Process;
```

Figure 9: WRK doxygen documentation - screenshot

`KeBatchGetNextThread` for the current batch. Depending on the batch policy and the contained threads the batch finishes execution and the scheduler can select the next item.

As shown the batch scheduling concept can be integrated into an existing round-robin operating system scheduler. By comparing an umodified and a modified kernel the concept can be evaluated.

3.3 Implementation with the Windows Research Kernel

3.3.1 Overview

In the summer of 2006, Microsoft released the Windows Research Kernel (WRK) which contains the kernel source code of the latest Windows operating system. The provided build-environment allows compiling modified kernel versions and then using these to boot a Windows Server 2003 Enterprise Edition machine.

Intended WRK usage is in lectures and other academic and research purposes [12]. Besides the source code, a free release of Virtual PC 2004, debugging tools, and the original design documents for Windows NT are contained in the WRK release.

The kernel source code is well documented, but for a better understanding of the kernel structure and source dependencies additional tools are helpful. Doxygen [9] is an open-source tool for creating different source code documentation representations (e.g. HTML pages) for different programming languages.

[9]http://sourceforge.net/projects/doxygen/

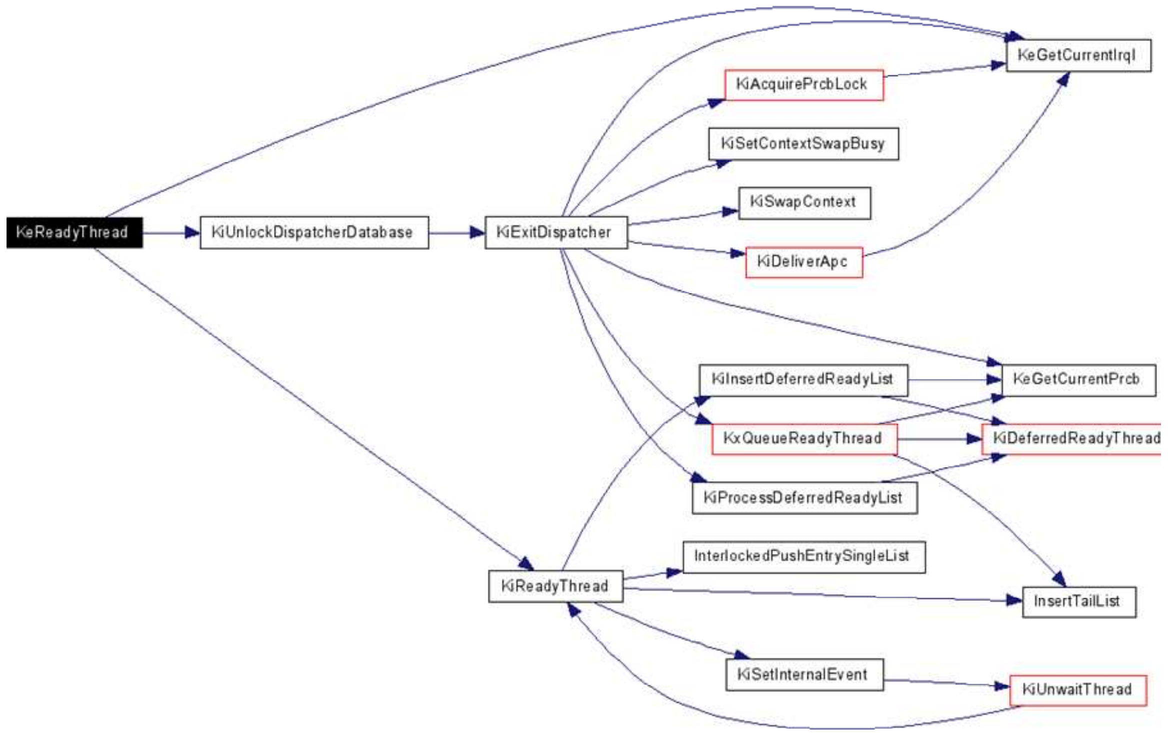


Figure 10: KeReadyThread callgraph

We created a source code filter which transforms the WRK documentation of a single C or ASM source file into the specific Doxygen format. In this way we can use Doxygen to create a detailed documentation for the Windows Research Kernel, including call graphs for functions. Figure 9 shows a screenshot of a Doxygen generated page, figure 10 shows an example of a created callgraph.

The WRK Doxygen documentation is hosted at the HPI [10]. Access can be granted to interested parties on request.

3.3.2 Implementation

The first step of adding new functionality to an existing operating system kernel is to insert a new system service call. One possibility is the usage of a kernel device driver which manipulates the system service call table. Another approach is the direct extension of the kernel. With the WRK this approach is possible for the Windows operating system.

The scheduler implementation in the Windows Research Kernel is distributed all over the kernel. The following list shows three important points where modifications are necessary.

- A new *batch datastructure* has to be introduced. This datastructure contains a linked list with assigned threads and configuration parameters describing the scheduling state and policy.

[10]https://dcl.hpi.uni-potsdam.de/wrk/

- The *thread datastructure* (file `ke.h`) must be extended with batch processing specific information. Basically a reference to a batch datastructure must be added. This reference points to the batch object if the specific thread is part of a batch.
- The different *Windows scheduling functions* must be modified in the way described above. Some examples: `KiSwapThread` is responsible for selecting the next thread for execution. `KiReadyThread` inserts a thread into the "ready" list. `KiQuantumEnd` is called if a thread has spent its time slice.

In addition to the modification of the round-robin scheduler, some of the batch scheduling policies described above have to be implemented. To investigate the applicability of these different policies, to develop more policies and to define guidelines when to use which version remains future work.

4 Conclusion and Future Work

Service-based systems make new demands to operating systems and application developers. A reliable execution combined with good performance is expected by clients and service providers.

A well adjusted operating system with a good concept for request processing, application isolation, memory management and activity scheduling can be the foundation for efficient and powerful service-based systems.

4.1 Expected contributions

Envisioned research results in the area of operating system abstractions for service-based systems will lead to an abstract model of services and service-based systems from an operating system point of view. Tools and operatings system extensions, which prove the applicability of the findings will be developed. Proposed research will use the "Windows Research Kernel" (WRK) as well as the Linux operating system as platforms for investigation of co-scheduling/batch-scheduling, resource pre-allocation and reservation, as well as the introduction of new, service-specific system calls and new application programming interfaces.

4.2 Next steps

The following list provides a roughly estimated time schedule of further work.

- Implement the batch scheduling concept with the Windows Research Kernel. Reimplement parts of the Apache webserver to use batches during request processing.
- Evaluate the new operating system kernel. First, overhead introduced by batch scheduling. Second, performance impact of batch scheduling.

- Investigate operating system support for application server. Monitor and measure server activities on operating system level. Compare with meta-information about service implementation.

References

[1] Thomas E. Anderson, Brian N. Bershad, Edward D. Lazowska, and Henry M. Levy. Scheduler activations: effective kernel support for the user-level management of parallelism. *ACM Trans. Comput. Syst.*, 10(1):53–79, 1992.

[2] Gaurav Bangs, Peter Druschel, and Jeffrey C. Mogul. Better operating system features for faster network servers. *SIGMETRICS Perform. Eval. Rev.*, 26(3):23–30, 1998.

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[4] Simon Beloglavec, Marjan Hericko, Matjaz B. Juric, and Ivan Rozman. Analysis of the limitations of multiple client handling in a java server environment. *SIGPLAN Not.*, 40(4):20–28, 2005.

[5] Michael Frisbie. A unified scheduling model for precise computation control. Master's thesis, University of Kansas, 2002.

[6] S. Harizopoulos and A. Ailamaki. Affinity scheduling in staged server architectures. Technical report, Carnegie Mellon University, March 2002.

[7] Poul-Henning Kamp and Robert Watson. Jails: Confining the omnipotent root. In *Second International System Administration and Networking Conference (SANE 2000)*, May 2000.

[8] James R. Larus and Michael Parkes. Using cohort scheduling to enhance server performance. In *LCTES/OM*, pages 182–187, 2002.

[9] Erich Nahum, Tsipora Barzilai, and Dilip D. Kandlur. Performance issues in www servers. *IEEE/ACM Trans. Netw.*, 10(1):2–11, 2002.

[10] Susanta Nanda and Tzi cker Chiueh. A survey on virtualization technologies. Technical report, Department of Computer Science, SUNY at Stony Brook, February 2005.

[11] M. P. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Commun. ACM*, 46(10):24–28, 2003.

[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] Daniel Price and Andrew Tucker. Solaris zones: Operating system support for consolidating commercial workloads. In *LISA '04: Eighteenth Systems Administration Conference*, pages 241–254. USENIX Association, November 2004.

[14] Mark E. Russinovich and David Solomon. *Microsoft Windows Internals*. Microsoft Press, 4th edition, 2005.

[15] Douglas Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects*. John Wiley & Sons, 2000.

[16] J. E. Smith and Ravi Nair. An overview of virtual machine architectures, 2004.

[17] Rob von Behren, Jeremy Condit, and Eric Brewer. Why events are a bad idea (for high-concurrency servers), May 2003.

[18] Matt Welsh and David Culler. Virtualization considered harmful: Os design directions for well-conditioned services. In *HOTOS '01: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, page 139, Washington, DC, USA, 2001. IEEE Computer Society.

[19] Matt Welsh, David Culler, and Eric Brewer. Seda: an architecture for well-conditioned, scalable internet services. *SIGOPS Oper. Syst. Rev.*, 35(5):230–243, 2001.

[20] Matt Welsh, Steven D. Gribble, Eric A. Brewer, and David Culler. A design framework for highly concurrent systems. Technical report, UC Berkeley CS, Berkeley, CA, USA, 2000.

A Task-oriented Approach to User-centered Design of Service-based Enterprise Applications

Matthias Uflacker
matthias.uflacker@hpi.uni-potsdam.de

This work positions the authors research activities within the field of service-oriented systems engineering and gives an overview on identified problems, related work and goals. The area closely investigated comprises the development of specialized, distributed, and flexible enterprise applications in combination with the effective implementation of usability enhancing design methods therein. Special consideration is taken into the distinct complexity of enterprise software systems and the resulting demands on development processes and design. The precise problem scenario addressed by the ongoing research work is settled in the area of engineering processes for interactive, service-based business process applications. A model-based approach to collect, formalize, and manage user-validated knowledge about roles, tasks, and interactions is proposed and related to user-centered design methodologies. Its potential to optimize the development process and support in the user and business value driven creation of innovative, composite enterprise software is discussed. The long-term goal of this work is to provide guidelines and tool-supported methods to establish a design-led and cost-effective engineering approach for service-based enterprise applications, emphasizing usability, user experience, and overall product satisfaction.

Keywords: Enterprise Applications, SOA, Web Services, User-Centered Design, Interaction Design, Model-Driven Development

1 Introduction

Carefully balancing desirability, feasibility, and viability of a product is the key for broad acceptance and market success. Nevertheless, user experience as a soft factor has been (and still is) neglected more than often in traditional software engineering processes, resulting in undesirable and inconvenient products [24]. User-centered design methodologies slowly find their way into software development processes, but projects are struggling with gathering, handling and applying user knowledge and the final transfer of results acquired during the design and prototyping cycles into established development phases [15]. It is obvious that a solid and planned ambition for innovative and usable enterprise software products still requires considerable adjustments in the software development process and support by appropriate tools.

The demand for investigating and introducing novel, design-led and outside-in driven processes (in contrast to traditional, inside-out development approaches), becomes more apparent when considering the advancements being made in software technology and the comparatively small changes that user interfaces have undergone in the meantime. In contrast to the many-fold increase in computational power and possibilities over the last centuries, user interface techniques remained essentially the same [3], but were applied to more and more complex scenarios. A design-driven and user-centered development process, focusing on end-user tasks and activities, can unfold new possibilities and innovative ways to better address system complexity on the front-end level and to optimize user interaction and overall user experience with the product.

Service orientation and service-oriented architectures (SOA), as a modern paradigm for developing and structuring agile software systems, provide an approach to combine loosely-coupled, distributed, and independent software services to a composite solution, thus making remote functionality accessible to other software components and applications. Following the '*software as a service*' principle, service orientation can help to better handle and manage system complexity by improving the flexibility and maintainability of software architectures and providing data and functionality "on demand". Nevertheless, very little support is provided for the design and creation of interactive applications that present service functionality and data to the end-users. If not to risk loosing flexibility and agility properties gained from the SOA philosophy, the development procedures for appropriate user interfaces of service-based solutions also need to support those attributes: Interaction design and UI modeling techniques have to incorporate interactions between users and services in order to enable a quick and seamless adaptation to changes in workflow and system requirements without much efforts in time and coding. Proper procedures and tool support for end-user knowledge management and application help in keeping costs, time-to-market and response times for changing requirements low.

We consider the service-oriented approach as a new opportunity to strengthen user awareness and task-orientation in the design process of interactive business applications. Information and functionality made available exactly where and when it is needed in the workflow, simple-to-use software that is fully adapted to the business and the end-users needs. Those are goals that can be realized by applying user-centered design methodologies in a service-oriented context. The creation of user interfaces for service-based systems must not be driven by technological specifications of software interfaces or service calls. More specifically, it has to be guided by an outside-in design process that focuses on activities [20], workflow knowledge and end-user validation. Empathy for the users and a profound understanding of their needs will lead to high-quality concepts for user interfaces that foster usability, satisfactions and productivity. The SOA paradigm is playing an enabling role in this approach as it may render previously unrealizable design concepts technically feasible.

Therefore, it is the purpose of this work to investigate the integration and adoption of usability engineering techniques in the context of interactive, service-based enterprise applications. This will include special consideration of modeling techniques for user tasks and user-service interactions. The question for which the proposed work wants to give an answer is ”How can the specification of workflows and user-service interactions help in the design and maintenance of service-based applications, improve the quality of user interfaces, and how can such specifications be modeled and integrated into a user-centered development cycle for enterprise applications?”.

2 Problem Definition

The diversity and complexity of today’s various business rules and processes constitute great demands on modern standard enterprise applications, developed to support in these processes. The strong heterogeneous and globally distributed landscape of targeted customers and users, tasks and scenarios, calls for highly capable, yet flexible and adaptive software solutions, that need to incorporate an umpteen number of often conflicting requirements: To compete on the market, the software has to be able to quickly react to changing business scenarios and tasks. Furthermore, it needs to offer a wide scope of functionality together with adequate integration capabilities for different platforms and existing system landscapes. Simultaneously, the user interface of the enterprise application itself has to retain flexible and adaptable to its individual user groups, their specific tasks and environments, in order to gain a high level of acceptance and applicability in targeted scenarios.

With the tremendous effect and influence on the productivity and the way people conduct their daily work, the end-user experience of enterprise applications is now a major differentiator in the software market. Still, application development was and is in this regard a relatively closed and sealed-off process in which usability methods and end-user involvement are rarely incorporated effectively [24]. But, a concentration on the end-users needs and wants is essential for future generations of business applications if they are to be designed to successfully address the growth in complexity and resulting user interaction problems. Otherwise, software developers are running the risk of creating tools which are difficult to learn and understand, and which are unsupportive and hindering the user in the execution of the business task. Through the substantial increase in software pervasiveness [5], usability has finally become a critical factor for the success of enterprise software, as it contributes to reducing errors and costs, as well as to increasing user productivity and satisfaction. Yet, profound knowledge on how to best adapt and apply methods to reach for an optimum in usability and user experience in enterprise applications throughout the entire development process is still missing.

2.1 Enterprise Application Development: Dealing with Complexity

As a consequence, modern sophisticated business applications like enterprise resource planning systems (ERP) are typically characterized by strong complexity in user interaction and user experience. This is provoked by the inherent complexity in data, functionality, architecture and configuration parameters, which again is a result of the steady increase in functional and non-functional system requirements, demanding business processes and challenging scenarios of use. The problem domain of enterprise applications is extensive and does not only lead to complex and bloated software products. It also renders the development processes itself much more difficult by enforcing increasingly complex programming and run-time environments, deployment infrastructures, and architectures [27]. Simultaneously, the growing amount of available data and implemented functionality results in more and more information that can be made available to the user. As such, human-computer interfaces of business applications are growing in complexity as well, impacting the usability of the product.

This is especially true in a mere technology driven development process, in which design decisions are typically based solely on feasibility factors and contracts and do not consider results of need-finding processes conducted with end-users. In such a scenario, user interfaces are likely to be designed to merely present an unfiltered view on the functional complexity of the software back-end. Such a procedure may result in software solutions that are hard to learn and understand, unusable, and unsupportive in the execution of certain business tasks.

It is a great challenge in enterprise application development (EAD) to simultaneously support a complex set of functions and business processes on the one hand, and to make the system easy to use in broad and differing application scenarios on the other. This has to be a major goal for enterprise application vendors if they want to differentiate from their competitors by designing their products usable, comprehensible, and desirable. Consequently, dealing with complexity in enterprise application development appropriately is a must.

Any profound discussion on design issues in EAD and business software requires a thorough comprehension of this distinct complexity of enterprise systems and their development processes. Neglecting the characteristic constraints and factors inherent in enterprise software projects will inevitably lead to solutions that do not appropriately address the specific requirements in product and development. Therefore, a detailed discussion and analysis of complexity problems in enterprise application development and products is conducted in chapter 4.

2.2 Striving for Simplicity

The intense complexity inherent in enterprise applications raises the discussion how user interfaces can be designed to hide the underlying business complexity at the best possible rate. The massive amount of data and functionality necessary and available in modern enterprise information systems makes it difficult for interaction designers to create a user interface that adheres to key usability attributes such as those defined by Nielsen [17], e.g. learnability, efficiency, memorability, and user satisfaction.

Simplicity in the design of the whole user experience and interaction with the system is considered as an important key factor for usable and desirable enterprise applications [18]. Simplicity is the key to provide a software tool that offers a wide range of functionality and enables sophisticated business processes, but at the same time is easy to learn, supportive and helpful in the fulfillment of tasks. Simple applications provide data and functionality only when and where it is required, reducing informational overload and front-end complexity. Thus, simplicity in user experience enables a seamless working process, ease of use, and a high level of product acceptance and satisfaction for the end-user.

Striving for simplicity is a continuous process of knowledge gathering and decision making, evaluation and selection [26]. What information is required where? What is the user exactly doing to fulfill her role and tasks? What information or functionality can be omitted? Being able to find the right answers to these questions during the development process requires deep understanding of the user as well of technological and business constraints. This knowledge can not be provided by a homogeneous team of software developers. Contrariwise, a multi-disciplinary design team, consisting of usability experts, user interface designer, business experts, and software technicians is necessary to strive for innovative ideas that render the application more useful and satisfying for all stakeholders.

But the challenges in coordinating and adjusting domain knowledge, communication between different experts, as well as the involvement of end-users in the requirements engineering phase and in iterative design evaluations, bears a number of problems and open questions especially in large software projects. Scalability, knowledge transfer and management, time and budget efforts are big issues when it comes to integrate multi-disciplinary and user-centered design methods in large and complex development processes. A careful coordination, selection and effective implementation of appropriate methods is essential for a successful approach towards developing simple, usable, and satisfying enterprise applications.

2.3 Enterprise Services and User Interaction

Web Services and related technical standards (WS-*) have found recent attention and application in the business software community. As a realization of the promising and lively debated service-oriented architecture paradigm, Web Service standards provide the chance to increase flexibility, adaptability, and maintainability of software systems by providing a communication infrastructure between decentralized, self-contained, interchangeable, and loosely-coupled software artifacts, described by open XML based interface specifications.

Enterprise Services are relatively coarse-grained Web Services that offer re-usable and dedicated functionality especially relevant for electronic processing of business data and workflow tasks in enterprises. As building blocks for specialized business solutions, Enterprise Services can be composed and interconnected to pass and receive data to and from several enterprise information systems (EIS) in order to achieve a targeted business objective. Today, software providers like SAP and IBM reorganize or extend their product portfolio accordingly, heading for service orientation by offering service-enabled business components and platforms. Applications that leverage and combine one or more software services in order to provide enhanced functionality or user experience are generally referred to as *composite applications* [23].

The specification and coordination of automatic service calls and information flows can be supported by business process models and business process management systems (BPMS). Such a *process-enabled SOA* [14] enables the specification and rapid adaptation of enterprise applications by means of formally describing business process models and promises agile adaptation to changing business requirements. Automatic service composition, service orchestration and choreography are subject of active research and upcoming state-of-the-art in the automation of business processes and service calls.

The execution of business processes, however, also involves human activity. People can take several roles in business process management and execution, like process stakeholder, owner, initiator or administrator. There is also a number of reasons for users to interact with services, such as the entry of required data, process initiation, suspension, escalation of conflicts, exception management, task nomination, or approval making. Neglecting human participation, user interaction, and the importance of high quality user interfaces for business process applications disobeys large parts of the problem domain.

A service is well designed if it puts focus on a delimited problem or purpose, but at the same time is re-usable and generally applicable as much as possible. This creates multiple scenarios in which a certain service can be employed. Depending on the application and user context, different service requirements and varying sets of relevant input and output parameters come to the fore. For example, an information service for a human resource system can be accessed by different employees via the enterprise portal. Depending on the role and position a user takes in the organization, he or she either has access to the complete set of employee data or only to a restricted subset of information (e.g. excluding work history and salary). In other cases, some input or output values of a service are simply irrelevant in a certain business context. In order to hide the functional complexity of the service and to deliver simplicity in user experience, available parameters and information should be reduced to a meaningful minimum in each context of use.

The problem lies in the identification of relevant data in the specific scenario. Similar issues are related to data entries for subsequent service calls and workflows, which users conduct in their environment. Some data might become available several steps before the actual service call takes place. In the meantime the data might be accumulated, used for other purposes, checked for consistency, or might be updated. Depending on the scenario, input and output parameters can be distributed over multiple steps in the user workflow. Thus, a simple request-response user interface, as it could easily be automatically generated based on service interface descriptions, is a naive solution for user-service interaction and most probably does not reflect the actual user workflow.

Nevertheless, there is no such formalism to describe interactions between users and services in a given application context.

3 Related Work

This chapter gives a brief introduction on various fields of research that are closely related to the proposed work described in this paper.

3.1 User-Centered Design

The idea of User-Centered Design (UCD) is motivated by the well-founded assumption that early involvement of end-users and constant evaluation and feedback sessions in the development process will significantly increase the quality of the product by meeting end-user requirements more appropriately and improving the overall usability of a system. The basic concepts of UCD in human-computer interaction go back to the work of Norman and Draper in 1986 [19]. The recently growing interest in user-centered design methodologies and questions on how to integrate the basic concepts into traditional and established development processes and teams arouses from demanding usability problems introduced by increasing software complexity, functional requirements and user experience demands over the past years.

Usability, as a key measure for the outcome of any user-centered design process, is defined by Nielsen as a composition of learnability, efficiency of use, memorability, errors, and subjective satisfaction [17]. A more formal and widely adopted standard definition of usability is ISO 9241-11 (Guidance on usability) [9], stating that usability is ”*the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*”.

A number of works exist that address and support the integration and improvement of usability enhancing methods in the development cycle of software products. Foundation for most of these, and standard to provide guidance in the design of usability, is published in ISO 13407 ’*Human-Centred Design Processes for Interactive Systems*’ [10]. Among the more prominent ones we find the often-cited approaches and methods published by Nielsen [17], Beyer and Holtzblatt [4], Jokela [12], Mayhew [16], or Vredenburg et al. [25].

The application of UCD techniques usually aims for the shift from a technology-driven ’*inside-out*’ development approach to a user-driven ’*outside-in*’ design thinking within the project, which is not limited to the mere design of user interfaces, but should consider the whole user experience of the product, including training, maintenance, and support. Early and constant user focus, a multi-disciplinary iterative design and steady evaluation of prototyped design solutions are the basic and characteristic attributes in reaching this goal. User involvement is a substantial part of UCD techniques and is commonly realized through interview sessions, on-site observations, participatory design, focus groups, or walkthroughs.

Prototyping is the central mechanism in UCD to gradually evaluate designs and development progress. The prototypes evolve from simple pen-and-paper prototypes

User-Centered Design	**Usage-Centered Design**
Focus on users: user experience, user satisfaction	Focus on usage: improved tools supporting task accomplishment
Driven by user input	Driven by models
Substantial user involvement: user studies, participatory design, user feedback, user testing	Selective user involvement: exploratory modeling, model validation, structured usability inspections
Descriptions of users, user characteristics	Models of user relationships with system
Design by iterative prototyping	Design by modeling
Varied, often informal or unspecified processes	Systematic, fully specified process
Evolution through trial-and-error	Derivation through engineering

Table 1: Differences between User-Centered Design and Usage-Centered Design (from [6])

in early phases to functional prototypes deployed on real systems. This low-cost approach in the beginning of user exploration allows for a rapid verification of user requirements and usability, following the maxim of *fail early, fail often, fail cheaply*.

3.2 Usage-Centered Design

Usage-Centered Design [8] is related to User-Centered Design in the sense that both techniques aim to improve the usability and utility of interactive software products. However, some significant differences exist in regard to the level of user involvement, knowledge description, and engineering process. Usage-Centered Design puts emphasis on user tasks and activities and tries to deploy a specified development process that guides in the user-validated implementation of those activities. Thereby, it heavily depends on closely related models that are responsible to capture user relationships with the system: *a role model capturing salient characteristics of relationships between users and a system, a task model representing the fine structure of work users need to accomplish with a system, and an interface model representing the contents and organization of the user interface needed to support the identified task* [6]. This model-driven process implies user involvement through early field investigations and usability inspections, in which derived knowledge is directly captured in the models and refined continously.

Usage-Centered Design aims to possess good integration and scalability capabilities regarding traditional software development processes. However, it does it at the expense of user involvement. Also, putting models in the center of process instead of design artifacts, bears open question related to user participation and validation, as end-users are usually not familiar with formal specifications and modeling techniques.

Table 1 captures the main differences between User-Centered Design and Usage-Centered Design.

3.3 Interaction Modeling

The specification of relations between users and systems takes fundamental part in the model-based development of user interfaces. Entities that form these relations usually consist of roles, use cases, tasks, and scenarios. A prominent representation of use cases is defined by Jacobsen et al. [11] as ”sequences of actions, including variant sequences and error sequences, that a system, subsystem, or class can perform by interacting with outside actors”. A modified and abstract form of these represent task cases or *essential use cases* by Constantine [7] and model ”the discrete intentions of users playing roles in relation to a system, taking the form of an interrelated collection of highly simplified narratives that are abstract, implementation independent, and devoid of technological assumptions”.

Task models represent the precise workflow a user or role executes to fulfill an identified task in a certain scenario. Most prominent in this area are ConcurTaskTrees (CTT) by Paterno [21]. CTT models diferentiate between user tasks (only performed by human actors), application tasks (completely executed by the software), interaction tasks (performed by the user interacting with the system), and abstract tasks, which are composed of several subtasks of previous kinds. For a detailed overview on task-oriented approaches and models, refer to [21].

UsiXML [1] (USer Interface eXtensible Markup Language) is a set of XML languages (schemata) to define user interfaces of interactive applications. UsiXML places emphasis on device independence, platform independence, and modality independence by partitioning interface characteristic into concept, abstract and concrete layers and by defining transformations and relationships between those layers.

4 Complexity in Enterprise Applications

Like in many other engineering disciplines, the development of an artifact (a software tool in this case) is bound to multiple rules, constraints and obligations imposed by project external or internal factors. Such a patchwork of requirements and constraints makes it such a challenging task to deliver a product as requested and within appointed time and budget limits. Especially in the design and implementation process for sophisticated enterprise applications like ERP systems we can identify aggravating, yet typical factors for complexity, which are characteristic for this engineering domain.

4.1 Classifying Complexity

It is not the purpose of this paper to present another quantitative or qualitative approach to assess software complexity. A number of different approaches for measuring and discussing software complexity has been analyzed for this purpose (e.g. [1], [13]). Rather, it is to show that high feature coverage and functionality support leads to an eminent increase of complexity in the back-end of interactive applications, which in turn breeds problems in user interaction design and usability.

To give a definition of software complexity we refer to [2], in which complexity is defined as a measure of the resources expended by a system while interacting with a piece of software to perform a given task. In [5], the authors add that ”*if the interacting system is a computer, then complexity is defined by the execution time and storage*

[1] http://www.usixml.org

required to perform the computation. If the interacting system is a programmer, then complexity is defined by the difficulty of performing tasks such as coding, debugging, testing, or modifying the software". We extend this definition by stating that if the interacting system is an end-user, then complexity is defined by the difficulty of performing a desired business task. End-users are confronted with application complexity on user interface level. This 'front-end complexity' relates contrary to a measure of user interface quality e.g. in terms of usability, ease of use, and learnability.

On back-end level, we can distinguish between intra-complexity, which denotes complexity within a single software module or component, and inter-complexity, which is caused by module or component inter-dependencies. We claim (and show) that a high level of back-end complexity tends to cause an immoderate level of user interface complexity on front-end side if not addressed carefully with appropriate methods in the development process.

The growth of structural complexity in software system does not surprise, since the increase in implemented functionality inescapably leads to an increase of code complexity and functional interdependencies. Software functionality has to be made accessible to the user or other software procedures. It requires a set of input/configuration parameters and produces a set of output values. From there we can deduce an inherent progression of user interface complexity: the more functionality is covered by an application, the more functions, input and output parameters have to be accessible for the users, including input data, results, error messages, warnings, etc. This rise in functional demands and software complexity not only complicates the development process in a whole but also renders the fulfillment of non-functional requirements like maintainability, extensibility, security and dependability more difficult. This again adds to the overall complexity of the product and affects the usability of applications. Nevertheless, in order to survive in a highly competitive market, software vendors are impelled to equip their applications with more and more functionality as demanded by customers or provided by competitors, leaving them no choice other than pushing the complexity of their products.

4.1.1 Functional Complexity

The functional complexity of a software product is determined by the size and extent of the problem domain that shall be addressed by the software. This includes the number and complexity of supported functions and business processes as well as the diversity of the targeted audience. It quickly becomes obvious that large enterprise software projects like ERP development have to deal with an enormous problem domain. Business processes steadily gain in complexity due to a massive increase of relevant and available business data, the number of involved participants, ongoing integration of multi-party processes, the inconstancy of business models, and the growing demand for decision support (real-time enterprise), ubiquitous systems, and anytime/anywhere computing. In this highly unsteady and distributed environment it is extremely challenging to provide a functional software framework that can be utilized and customized to fit the needs of a wide range of globally distributed potential customers of different size, organizational structure, coming from diverse industry branches. This not only results in an extensive set of different user types, but also leads to a complex mesh of country and industry specific regulations and laws (e.g. tax, accounting and reporting directives, Sarbanes-Oxley Act) that have to be observed and supported by the tool.

This heterogeneity of the customer landscape is inevitably augmenting the internal functional complexity of a software component as it exposes a number of special cases, singular requirements, and separate treatments for each targeted customer group. Additionally, the consolidation and automation of business processes effects an increase in inter-component complexity. Processes more and more depend on and affect external data and span multiple business components. For example, an ERP sales component for order management might bear relations to stock and production components for automated availability checks and to a finance component for booking and accounting purposes.

4.1.2 Non-functional Complexity

The IT landscape of a business organization is rarely built up from scratch. More often, several parts of legacy systems are replaced by new implementations or single layers of multi-tier architectures are redesigned or extended. The development team who is (re-)engineering parts of a software system has to ensure compatibility with existing and collaborating hard- and software resources. This landscape integration imposes several constraints on the development process and often complicates the technical realization. An existing code and data base, heterogeneous legacy systems or predetermined UI technologies often demand for software adapters and workarounds.

Due to its economical value and critical importance for business organizations, enterprise software has to comply with stipulated quality requirements. Ranked differently from project to project, fundamental exit criteria usually comprise nun-functional requirements like robustness, security, performance, maintainability, extensibility, or, often of minor importance, usability. Fulfilling those requirements can add to the degree of complexity as it postulates careful architecture design and technology consideration (e.g. incorporation of monitoring techniques, security tests and so on). Nevertheless, it is common knowledge that in order to compete in the steady race to market, projects are regularly forced to reduce time and costs required for designing the software in regard to those requirements. Such a proceeding may not only compromise the overall quality of the software as it increases the chance of creating flawed code, but again is likely to add complexity to the final product e.g. by reducing the separation of concerns on the back-end and disregarding interaction design on front-end side.

This effect is often aggravated by large heterogeneous and possibly distributed development teams, different mindsets, organizational hierarchies, decision policies, as well as strict time-to-market and budget obligations.

As a result of strict and comprehensive non-functional requirements in combination with an exceptionally complex problem domain, standard enterprise solutions like ERP and workflow systems consist of a noteworthy complexity level which requires special and careful consideration during the design and engineering process.

4.2 Case Study: R/3 Sales & Distribution

In order to be able to analyze and evaluate ways and methods to improve end-user experience in large-scale enterprise applications, it is important to grasp and understand the inherent nature of complexity in such systems. Thus, the objective of this study is to exemplify enterprise application complexity by an in-depth analysis of sales order management and order variations in SAP® R/3® Sales & Distribution (SD) module. We investigate the origin and implementation of different customer requirements and deduce therefrom the need for the copious number of supported business functionality and parameters found in the software. By this means, this work helps in better comprehending and assessing reasons for certain design decisions and to evaluate and identify potential for design optimizations. Taking an example from within the area of standard ERP software is founded in the high level of expected complexity and well-known usability issues in those products.

4.2.1 Overview

R/3 as a fully-fledged and well-established standard ERP system was SAP's integrated business solution for large-scale enterprises until the release of its successor mySAP ERP. Still very commonly in use, R/3 features a holistic and integrated approach for a wide range of different business tasks. R/3s extensive range of functionality is delivered by a set of collaborating, but basically independent modules, of which each one is responsible for a delimited business area. Besides the Sales & Distribution module, which has been chosen for closer investigation in this chapter, we find among the most widely used modules e.g. Financial Accounting (FI), Materials Management (MM), Controlling (CO), Production Planning (PP) and Human Resources (HR).

By leveraging functionality and by sharing information in between modules, this approach provides a high level of business integrity. Here, business integrity is itself composed of data integrity and process integrity [22]. Data integrity constitutes the shared use of enterprise master data by multiple functions from different modules. Process integrity originates from function calls and data transfer between modules, allowing for a seamless and automated process flow across the enterprise. Taking an order-to-cash scenario as an example, an integrated process might affect functionality and data related to order management, material master, transport & delivery, and financial accounting. Despite the modularization of the software architecture, data and process integrity again supports increased system complexity by creating inter-component dependencies.

R/3 reaches a high level of variability and flexibility by allowing customers to select from available modules only those which are required in their specific business scenario and to customize the software to their needs. The system grows with the enterprise as further modules and resources can be installed and configured as needed. With the provision of highly configurable and interoperable business functionality, R/3 tries to fully cover the problem domain of enterprise applications. Such a great degree of customizability and adaptability of course, has its downsides. The total cost of ownership and deployment time is considerably increased by high installation efforts, maintenance and configuration overhead.

4.2.2 Customer Landscape

Due to its flexible and generic nature, R/3 is not restricted to be applied by customers in a limited area of certain industries and branches, but provides solutions for a wide range of business cases. Nevertheless, a classification of industries addressed by R/3 makes sense, as single modules and functions can be adopted based on best practices and special requirements in those areas. R/3 industries are divided into 28 top-level sectors (Fig. 1) which again can be subdivided into more fine-grained branches.

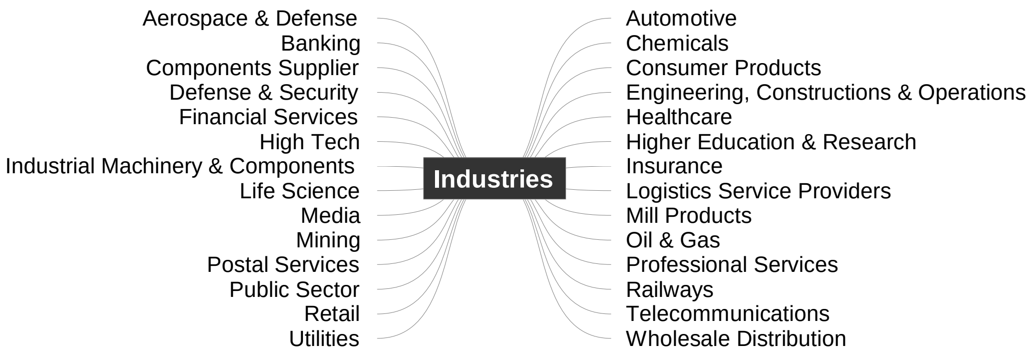


Figure 1: Industries addressed by R/3

Anyhow, this classification does not imply that customers belonging to one and the same industry branch possess the same business requirements and processes. Differences in company size, organizational structure, produced goods or services require special consideration and precise configuration. Furthermore, given the fact that the targeted market for R/3 is to a large extent globally distributed and under influence of diverse conditions, laws and regulations, the heterogeneity of the customer landscape is pushed to the extreme. Capturing this whole diversity of functional and non-functional customer requirements with only one product is the challenge R/3 is bound to. Continuing the principle of modularization, R/3 addresses this challenge by supporting business' best practices in core modules and implementing specific and isolated branch functionality in Industry Specific (IS) modules operating on top.

4.2.3 Process Automation

It is, with no doubt, the automatic execution and combination of available business process functionality and tasks which leverages the full power of ERP systems. By enabling a high level of process integrity, an accurately customized R/3 system is significantly helping the user in its task execution and speeding up processes by doing background calculations based on configuration and master data. Typical sales processes which are commonly automated and executed when required (triggered by user, triggered by data becoming available) are availability checks for ordered items, price determination based on customer data, determination of transport and delivery details (fees, routes, etc), or accounting related processes (see fig. 2 for further examples). It is obvious that with a rise in process integrity and automation the level of interdependencies between system modules increases alongside. This becomes especially apparent when taking availability checks as an example. Here, data located in diverse sources has to be considered such as material master, storage and warehouse information, production plans or procurement orders.

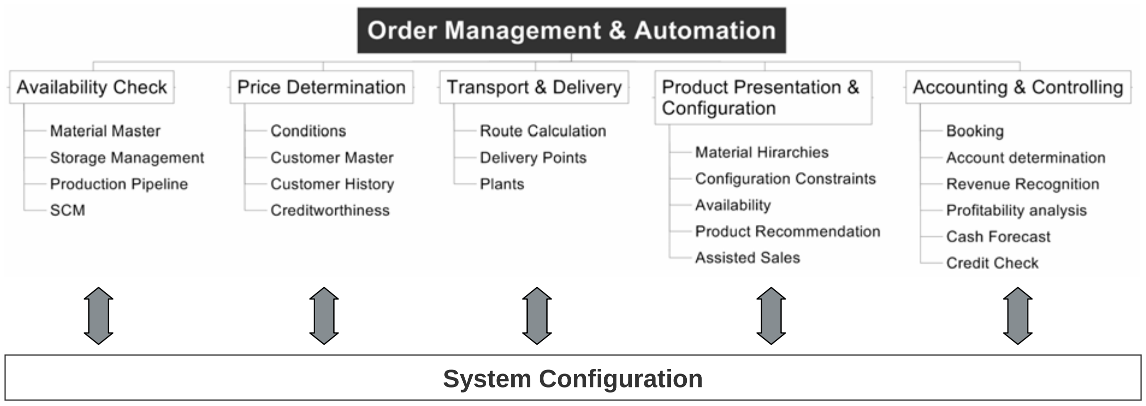


Figure 2: Process automation in SD

This extensive potential for process support, automation, and system adaptability ultimately contributes to the overall economical utility of the software and to the productivity of its users. Nevertheless, it has to be pointed out that the gain in system complexity is serious. Resources (time, money, IT infrastructure) and expert knowledge is necessary to carry out the required system configuration, customization and maintenance. Anyhow, these costs are often justifiably considering the resulting advantages.

4.2.4 Distribution channels and Order variants

Customer orders are handled differently to smaller or greater extent in each sales organization. This not only depends on who is ordering what, but is also affected by the degree of process automation and controlled by the system configuration of the R/3 installation. For each order scenario in a sales organization the system provides an order variant (order type) which is specifically configured to initialize and control the ordering process according to the requirements of the organizational unit. Most typical and common scenarios include orders for items that are delivered directly from stock (sales-from-stock), which are produced specifically for that order (make-to-order), or are configured specifically to the customer's own desire (configure-to-order). See figure 3 for a small selection of order scenarios found in SD installations.

Another important factor determining the ordering process is the customer who is placing the order. Whether the order (respectively the delivery address) is domestic, international or the order is placed internally by another company division, can have effect on price, tax and delivery calculations. Diverse information from customer master

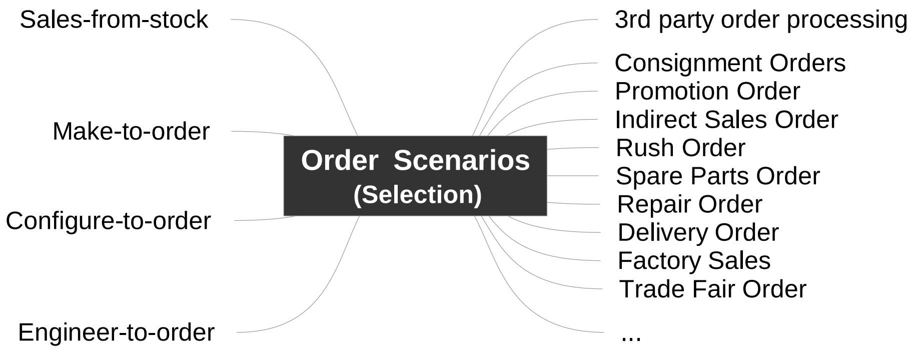


Figure 3: Typical SD order scenarios

data is used to determine pricing conditions, discounts, credit worthiness, etc., based on process configuration (automation) and order scenario specification.

Resulting from the diversity of the customer landscape, the R/3 SD module also has to cope with an equally complex diversity in materials that can be ordered and managed. This leads to an extremely bloated set of parameters that are attached to an order and to order items. An order can be divided into header data, order data and a list of items ordered. While data in header and order fields is relatively limited to holding information such as ordering party, payment and delivery details, the parameters available to each ordered item are much more diverse than the minimal set of material ID and order quantity.

We identified more than 70 parameters related to order items in the Sales & Distribution module. Each of these parameters had right to exist as it was required in at least one specific order scenario or used by a possibly very limited number of customers. Most of the parameters are only useful for certain industries and product types (e.g. Dangerous Goods Profile, Batch ID, Product Hierarchy, Bill of Material). Others are relevant only if the order management process is integrated and configured to automatically connect to business functions such as booking and account determination (e.g. Account Assignment Group) or supply chain management (e.g. Available-to-Promise Quantity).

4.3 Dealing with Complexity

A standard business solution like R/3, targeting a huge variety of customers and business tasks, has to provide an extensive set of functionality in order to proof applicability in various numbers of different scenarios. Providing the full functionality to all users and scenarios would result in a bulked and unusable user interface which is too hard to learn and to understand. Consequently, the massive complexity in the back-end has to be addressed in way which allows the software to be adjusted and fine-tuned to meet the specific requirements and demands of its customers and end-users. Customization and system configuration are appropriate approaches to make a product flexible, universally applicable and at the same time aligned to specific customer requirements.

Still, striving for simplicity in user interfaces has to be an important part in the pro-

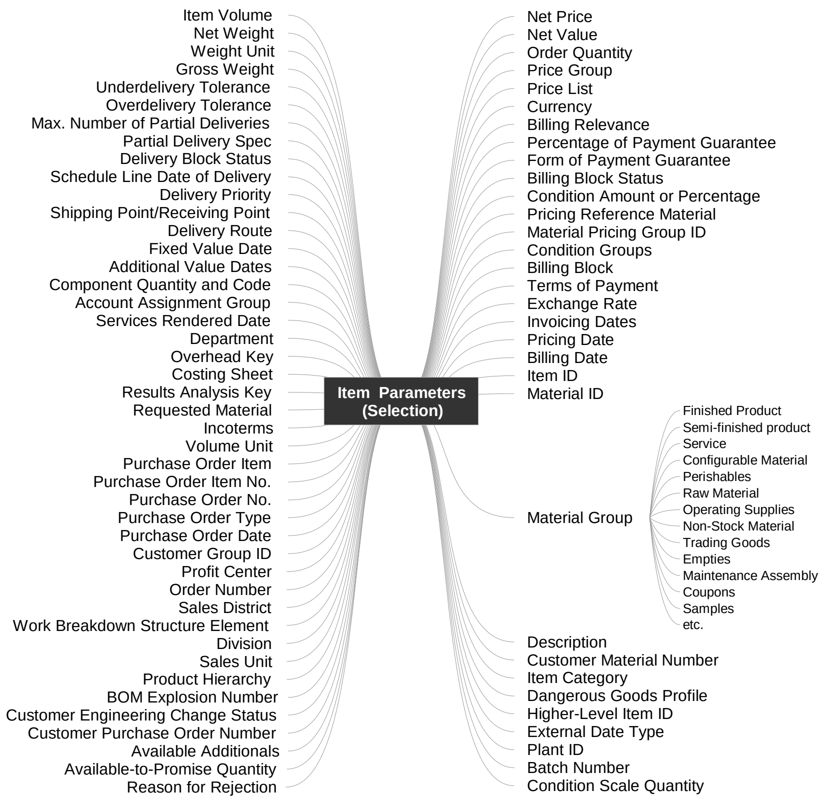


Figure 4: SD order item parameters (selection)

cess of designing user interaction. Otherwise, the large complexity of the software would inevitably lead to unsupportive user interfaces and diminish usability and end-user experience. Customizability alone is not sufficient to fully target the needs of individual end-users, as the possibilities might proof to be too restrictive or can not be influenced by end-users.

4.3.1 User-centered Interaction Design and Redesign

Addressing this complexity is always a process of finding compromises and trade-offs between contradicting requirements. To create the perfect application with minimal investment in time and budget is not possible, as there are always differences between what is desired and what can be realized under given constraints. To find the right balance and optimal solution for customer and end-users requires a good understanding of how the software is going to be used later on. Therefore, design decisions must not be solely based on functional requirements but need to embrace end-user knowledge and a thorough task analysis.

Several methods and patterns are known in user interaction design to help addressing software complexity and information overload, such as progressive disclosure (information hiding) or guided activities. User involvement is not only important when deciding on what methods to apply but also essential when asking how they should be applied best. For example, progressive disclosure as a method to reduce information displayed on the screen and to show this information when requested helps inexperienced users to work with the application but reduces efficiency of expert users by increasing the amount of interaction (e.g. mouse clicks) required to acquire certain information.

4.3.2 Reduction by Focusing

An alternative approach to reducing complexity of enterprise applications is to focus on a restricted customer landscape. To deviate from the principle of supporting a maximum number of business scenarios and different customer requirements is a design decision with significant impact. To focus means to reduce the scope of the software and to explicitly target a well-defined but smaller market. This may imply to concentrate on standard solutions specifically designed to meet the requirements of one single industry branch or companies in specific countries and specific size. Obviously, the total number of business processes and special requirements that have to be considered and addressed by the software will reduce accordingly to the level of focus, ultimately reducing complexity and leading to product simplification.

Further simplification can be achieved by eliminating fields and parameters which are used very infrequently. But, decisions on what requirements have to be supported and what functionality can be eluded have to be made carefully, only after having researched and identified the demands that are specific and essential for the focused market. End-user research is an important and required practice to appropriately address the targeted market and to provide a solution which is generally applicable and useful in its defined domain.

5 Designing Service-based Enterprise Applications

As shown in the previous chapter, system complexity is a predominant reason for usability problems and low end-user acceptance, causing complex and hard to understand user interfaces and informational overload. It is obvious that a development process solely driven by technology and business requirements can not address these problems sufficiently. The approach under investigation is essentially focusing on user requirements and task exploration in the early design phase that is conducted and arranged with all participating stakeholders, including end-users, usability experts, technical experts, and business analysts. It combines best-of-breed aspects of User-Centered Design, Usage-Centered Design as well as interaction design techniques, and integrates those in a service-oriented software development context. Knowledge acquired in user studies and exploration phases is gradually transfered and formalized into a set of models that support in managing findings, developing prototypes for evaluation with end-users, team communication, and the final transfer of a design master into development cycles.

This model-based development methodology comprises existing techniques to formalize user roles, use cases, workflows, and scenarios, adapting and enhancing them accordingly to specific needs where required. It also introduces a *User-Service Interaction Model* as a new layer of abstraction to connect and map task knowledge and findings acquired in UCD-driven analysis phases to data objects and services used in the application. The model defines a temporal relationship between information and functional entities that are made available to the user, specifying which service functionality, parameters, and results are utilized at which position in the workflow. The next step towards UI concretion is a technology independent description of possibly multi-modal user interfaces which considers context and scenario of the tasks. The user-service interaction model gives valuable input for this high-level UI specification, as screens, windows and interactive elements are defined and partitioned accordingly.

Thought-out tool support and integration in the design process is essential to render the emerging modeling efforts marginal. The work of designers and the creativity aspect of user-centered design must not be controlled and influenced by this more formal approach, thus the modeling has to be as transparent and non-intrusive as much as possible. A collaborative work environment for collecting and managing knowledge and integration into an IDE can help to provide model visualization and warnings, e.g. when model discrepancies are detected. It can also support in the rapid development of prototypes for different roles and scenarios, as the knowledge retrieved from the models can be leveraged to automatically generate basic UI skeletons. Early and informal input techniques like pen-based digital sketching are promising approaches to support in the user-centered design process, simultaneously capturing knowledge for later refinement.

The underlying model stack can help in the development phase to truly identify and communicate end-users needs across the design team and to evaluate the correct translation into a service-based solution. Communication support is essential in a multi-disciplinary team, as design decisions have to made in consultation with business experts and code developers. The constant correlation and focus on user tasks and workflows results in a software product that is consistent with the conceptual model of the end-user, emphasizing learnability and utility.

While this approach primarily provides decision support and guidance during design time, it also has significant benefits in the later life-cycle of the application. Information on service utilization and call statistics can be accumulated and extracted from the models, either by task, roles, or scenario. Semantically enriched models could also support in semi-automatic service discovery and selection.

Thus, this approach appears promising to address the outlined problems in developing service-based enterprise applications and to support multi-disciplinary teams to design a solution that fulfills manifold requirements arousing from the business, technical, and user domain. Rapid development of prototypes, consideration of interaction patterns, a common base for communicating and managing design knowledge lead to a more effective development process, while the formal representation of user-service interactions allows faster customization and adaptation of high-quality user interfaces. Combined, we are able to create interactive service-based software solutions that are usable and helpful in the execution of working tasks and can react flexible to changing business needs: the next step to deliver software that fulfills the SOA aspirations.

6 Next Steps

Having identified and justified the need to adjust modeling and design techniques in the development process of service-based enterprise applications, future research work will comprise the following next steps.

A set of appropriate models needs to be defined alongside with interrelationships between them, which together meet the envisioned requirements for a design-driven, end-user- and task-oriented development approach. This includes the evaluation of applicability for existing models within this context and the definition and integration of a model layer for specifying user-service interactions. Further research will also address interaction patterns for user tasks and workflows that include communication activities with web services.

Next, these concepts will be related and embedded into a multi-disciplinary development environment to give a holistic perspective on the design of service-based software solutions. A prototypical implementation providing integrated tool-support in the modelling process for user-service interactions is planned to show the usefulness of such an approach and to evaluate it together with designers and developers in real-world scenarios.

References

[1] Mohsen AlSharif, Walter P. Bond, and Turky Al-Otaiby. Assessing the complexity of software architecture. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pages 98–103, New York, NY, USA, 2004. ACM Press.

[2] V.R. Basili. Qualitative software complexity models: A summary. Los Alamitos, Calif., 1980. IEEE Computer Society Press.

[3] Michel Beaudouin-Lafon. Designing interaction, not interfaces. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 15–22, New York, NY, USA, 2004. ACM Press.

[4] Hugh Beyer and Karen Holtzblatt. *Contextual design: defining customer-centered systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[5] Presseinformation BITKOM. Technologische Modernisierung. `http://www.bitkom.org/de/presse/8477_41261.aspx`, September 2006.

[6] Larry Constantine and Helmut Windl. Usage-Centered Design: Scalability and Integration with Software Engineering. In C. Stephanidis and J. Jacko, editors, *Human-Computer Interaction: Theory and Practice. Proceedings of the 10th International Conference on Human-Computer Interaction*, Mahwah, New Jersey, 2003. Lawrence Erlbaum Associates.

[7] Larry L. Constantine. Essential modeling: use cases for user interfaces. *interactions*, 2(2):34–46, 1995.

[8] Larry L. Constantine and Lucy A.D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley Professional, 1999.

[9] ISO/IEC. 9241-11 Ergonomic requirements for office work with visual display terminals (vdt) – part 11: Guidance on usability, 1998.

[10] ISO/IEC. 13407 Human-Centred Design Processes for Interactive Systems, 1999.

[11] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Professional, 1999.

[12] Timo Jokela. Making user-centred design common sense: striving for an unambiguous and communicative UCD process model. In *NordiCHI '02: Proceedings of the second Nordic conference on Human-computer interaction*, pages 19–26, New York, NY, USA, 2002. ACM Press.

[13] Joseph P. Kearney, Robert L. Sedlmeyer, William B. Thompson, Michael A. Gray, and Michael A. Adler. Software complexity measurement. *Commun. ACM*, 29(11):1044–1050, 1986.

[14] Dirk Krafzig, Karl Blanke, and Dirk Slama. *Enterprise SOA. Service Oriented Architecture Best Practices*. Prentice Hall, 2004.

[15] Ji-Ye Mao, Karel Vredenburg, Paul W. Smith, and Tom Carey. The state of user-centered design practice. *Commun. ACM*, 48(3):105–109, 2005.

[16] Deborah J. Mayhew. *The usability engineering lifecycle: a practitioner's handbook for user interface design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[17] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[18] Jakob Nielsen. *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing, Thousand Oaks, CA, USA, 1999.

[19] Donald Norman and Stephen Draper, editors. *User Centered System Design: New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[20] Donald A. Norman. Human-centered design considered harmful. *ACM interactions*, 12(4):14–19, 2005.

[21] Fabio Paterno. *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2000.

[22] Susanne Patig. *SAP® R/3® am Beispiel erklärt*. Peter Lang GmbH, Europäischer Verlag der Wissenschaften, Frankfurt a.M., 2003.

[23] Marcel Seelig. Performance considerations on composite applications. *13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS'06)*, pages 445–452, 2006.

[24] Giorgio Venturi and Jimmy Troost. Survey on the UCD integration in the industry. In *NordiCHI '04: Proceedings of the third Nordic conference on Human-computer interaction*, pages 449–452, New York, NY, USA, 2004. ACM Press.

[25] Karel Vredenburg, Scott Isensee, and Carol Righi. *User-Centered Design: An Integrated Approach*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[26] Carl Zetie. The Power Of Simplicity In Application Development. Forrester Research, Inc., December 2005.

[27] Carl Zetie and Liz Barnett. Why enterprise application development is so hard - and how it must get easier. Forrester Research, Inc., August 2004.

A Framework for Adaptive Transport in Service-Oriented Systems based on Performance Prediction

Flavius Copaciu
flavius.copaciu@hpi.uni-potsdam.de

Services are the newcomer in corporate networks and on the Internet, competing for resources alongside classical applications. While plenty of research has been done on SOA, most of the work has been concentrated at application level, the communication stack receiving less attention. This work tries to fill in the missing parts of the picture, providing an overview of the protocols used in todays SOAs and exploring some new possibilities. For doing performance modeling, evaluation and prediction of the protocol stacks and its components FMC-QE, is the proposed methodology. As a way to improve services, ATrA - an adaptive transport architecture is proposed. ATrA enables service consumers to dynamically switch between different transport stacks and select certain protocol options, according to a set of server and client side data. ATrA has a reasoning engine that makes use of the available data gathered from the client and server in order to dynamically select a transport stack that is considered to be optimal for the given usage case. The evaluation of Web Services invocation over UMTS is presented as a case study, where we consider that such quantitative evaluation is appropriate and an adaptive transport stack could improve the performance of web services.

1 Introduction

Service-oriented systems, implemented usually as web services and build using SOAP or REST, have received a lot of attention lately from both the industry and academic community. Key industry players such as IBM, SAP, Microsoft, Sun and others have positioned themselves strongly as front runners in the field of service-oriented computing. But with every company's marketing teams avidly looking for the new buzzwords, that will enable one to set himself aside from the IT crowd and hopefully gain a competitive advantage out of it, one has to wonder to himself what is hidden behind such concept. While SOA is clearly not the universal solution to all IT problems, when companies like SAP develop and build applications based on thousand of web services, when Google, Amazon or eBay enable access to their own business functionality over web services, it becomes obvious that web services are here to stay.

There are many definitions of the term services and web services, the one definition that has the highest relevance in our study of the transport infrastructure for web services is: services are software entities that are remotely accessible using XML based messages [29]. Most of the web services also have a WSDL description, but adding this requirement in the definition would disqualify the REST style web services. It is also important to notice that by this definition, services are independent of the transport infrastructure.

Communication systems, together with XML are one of the key components that enable service-oriented computing. Services, the fundamental building blocks of SOA, have to be made available over the network so that remote clients will be able to access them in order to consume them. A 'perfect network' is assumed in many cases when modeling, analyzing or deploying web services and this usually leads to difficulties and unwanted or unexpected behaviors of the whole system. Most of the fallacies of distributed computing still apply to today's networks and special care must be taken in order to avoid them. Usually this means choosing the protocol stack that offers the necessary functionality to counter them.

In many cases the same functionality can be achieved at different levels in the protocol stack, each option having its own advantages and disadvantages (speed, processing or memory requirements, load on the service provider, etc). The typical stack for web services makes use of the SOAP binding over HTTP and by this is taking away choices from the developer while providing most of the times a suboptimal solution. In many cases, the simplicity of HTTP simply outweighs the complexity of introducing an additional transport layer but this is not always the case. As a solution to the performance penalties, alternative bindings have been developed. At the same time, the supporting frameworks for web services have also evolved and now most of them offer the possibility to make web services available under multiple transport bindings.

However, this has not solved the above mentioned problem. It has simply added the task of determining the best binding to the application developer and generated new questions: is the binding still optimal for different client or server scenarios, what happens when using alternative service providers, etc. We believe that the programmer should not have to be concerned with the selection of one specific protocol stack for the service-oriented application. The supporting framework should perform this task, based on multiple data sources and including a usage profile that might be provided by the developer. The chosen protocol stack should be functional (both the web service consumer and provider should support it), adaptive (should change as response to new information like new protocol stacks available, changing environment conditions) and optimal (should be the best one taking into account existing information).

Achieving such desiderata is possible in our opinion and this work focuses on the first steps on this long road.

Section 2 gives an overview of the communication stack used in Service-Oriented Architectures and presents the most important communication protocols used by SOAs. For each of the protocols, the relevant state of the art is presented and a discussion investigates the protocols advantages and disadvantages in a SOA. Section 3 presents our proposed approach for doing performance analysis and estimation on the communication stack followed by section 4 where an architecture for adaptive transport in SOA is presented. The architecture design along with the required modifications on the service broker, provider and consumer are discussed. Section 5 presents the current status of a case study, where we consider that such quantitative evaluation is appropriate and an adaptive transport stack could improve the performance of web services over UMTS links. Lastly, section 6 gives an overview of the work and the conclusions that have been reached.

2 Overview of the SOA Communication Stack

2.1 SOAP

SOAP [2] is a lightweight protocol that provides an XML based framework for message exchange in a distributed environment [15] that has emerged as the standard protocol used for web service communication. SOAP does not specify a transport protocol but can make use of different available protocols for whom bindings have been defined. HTTP is the most used transport protocol for SOAP, in part because it makes it easier to interconnect systems.

The SOAP protocol is the center of a whole collection of W3C recommendations concerning web services. Many of the recommendations are focused on improving and providing additional features for SOAP. Some features, for example WS-Security, duplicate functionality that can be provided by the transport bindings used for SOAP. This opens the way to moving functionality to the SOAP level and might provide additional incentive for using lighter transport bindings.

SOAP based systems suffer two types of problems: on one side, the size of the SOAP message is quite big relative to the data payload and processing the XML document is a resource intensive task. The SOAP protocol and toolkits have been under a lot of scrutiny and a number of research papers have investigated different aspects of them. [15] is one of the most complete researches, investigating many aspect of a SOAP system: generating HTTP headers (if HTTP is the preferred binding), generating and parsing the XML document, SOAP compression as well as threading and scheduling issues in the web service container.

[28] presents a system where the SOAP payload is compressed and a solution that allows servers to dynamically adapt to the clients requesting compressed responses based on the current CPU utilization. The client is implemented on a handheld device and experiments are performed over several networks: wireless LAN, Bluetooth and an emulated GPRS network. The response time of the server as well as the number of connections per second that it serves are investigated.

Another approach to the size problem presented by SOAP messages is to use an alternate XML serialization, usually referred to as binary XML. There is strong opposition in the community to any changes that would make XML anything else but text, but the advantages of such a change are fueling the efforts supporting it. Binary XML encoding can also improve the parsing and querying time of XML documents. An extensive overview of binary XML encodings along with performance evaluations for a set of XML processing tools like TurboXPath, Xalan and Xerces (both Java and C++ versions) are presented in [4]. The results show that binary XML encodings can be parsed over 2 times faster then XML and that the choice of parser has strong influence on the performance of the whole system.

In conclusion we can say that SOAP offers many possibilities for implementing message exchange and that research looking at solutions for the most evident problems is well under way.

2.2 REST

While not a protocol, Representational State Transfer (REST) is a software architectural style for distributed hypermedia systems like the World Wide Web. In his own words, Roy Fielding, the person who designed REST, says: “Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use” [14].

REST makes use of resources or machine equivalents of noun, uniquely identified by URLs. Resources are concepts that can have zero, one or multiple representations. If no representation exists for a given resource it is considered that the resource does not exist. The clients need to be able to understand the representation of the resource, typically HTML, XML or another MIME defined type. The components of the network can interact with the representations by using verbs. These representations can be exchanged between components of the network using HTTP. The verbs defined by HTTP are: GET, PUT, POST and DELETE.

Together with SOAP web services, REST style web services form the majority of web service implementations available today. If we ignore the architectural principles that make them different, when considering the transport infrastructure, we can make several observations:

- most messages exchanged are in XML defined by a schema, but this is by no means a mandatory condition
- interfaces are limited to HTTP. The usage of HTTP verbs makes it impossible to consider alternative transport bindings.

REST style web services are a simple and effective way of implementing web services. The only requirements are HTTP and XML processing technologies that are widely available. The drawback is the mandatory use of HTTP which may be enough for many cases but might prove insufficient in more complex situations.

2.3 HTTP

HTTP is the most popular binding for the SOAP protocol. HTTP performance has been measured and modeled in a considerable number of research papers. When used as a transport for web services, the following aspects regarding HTTP are relevant: header building, compression, persistent connections and secure HTTP.

When compared with the TCP binding, the building of HTTP header and time required to send it seems to be the only difference. While the data overhead influence can be decreased by using a larger message size, the penalty imposed by the need to build the headers stays. HTTP 1.0 requires that the size of the payload (the SOAP message) be specified in the header. This means that the header can only be created after creating the SOAP message. After the header has been created, the two can be concatenated or sent via two different socket calls. HTTP 1.1 addresses this problem by providing support for chunked encoding of messages and requiring chunk to be proceeded by its own size. Both the HTTP1.0 as well as HTTP1.1 approaches present performance concerns.

On the fly HTTP compression can be used in order to reduce the size of the SOAP message to sizes comparable to the size of the original binary data. However HTTP gzip compression is computationally expensive and it may exceed the time required to actually send the uncompress data over the network [15]. For small devices as well as busy servers these processing requirements as well as the memory requirements have to be taken into account. The research done by [20] shows that gzip manages to compress the data by approximatively 40% for the cases considered. As outlined in [19], for small SOAP messages the generic compression mechanisms do not perform very well. However, if reducing as much as possible the amount of data sent over the network is desired (for example due to high data trensmission costs), HTTP compression might prove as a viable alternative that does not pose the incompatibility threat of alternative methods, like binary XML.

The use of HTTP 1.1 persistent connections is another possible way of improving web service performance by eliminating the overhead of TCP connection establishment. This might only prove useful if a service or services hosted by the same web container are invoked multiple times. When using persistent connections the server has to keep the connections open for a longer period of time and this might lead to security problems.

When the data exchange between the web service consumer and provider needs to be encrypted, HTTPS is a good option. The web service frameworks offering the HTTP binding usually implement HTTPS as well. While being very easy to enable encryption, the performance hit on the system has to be taken into account and the decision to use encryptions needs to be well assessed.

2.4 SMTP

Another SOAP binding often implemented by SOAP frameworks is the SMTP binding [24]. The SMTP binding permits the transport of SOAP messages as body of the SMTP message or as attachments. This binding is used mostly for asynchronous operations and is designed to make use of the existing emailing infrastructure. An interesting option when using the SMTP binding is the possibility to use existing standards and applications (S/MIME or the PGP based applications) in order to provide encryption or security.

2.5 TCP

The SOAP over HTTP binding imposes additional burdens regarding the transport of the SOAP messages, generating and parsing HTTP messages requires processing resources and the HTTP header increases the amount of data to be sent over the network. The SOAP binding only utilizes a small part of the HTTP capabilities while taking the full burden of the protocol. As specified in [2], the most important characteristic of HTTP that the binding makes use of is the correlation of request and response messages.

While not being an official SOAP binding, SOAP over TCP is already provided as an alternative binding by some providers, ex. Apache Axis[1] and Microsoft WSE[2]. SOAP over TCP implementations require less memory and processing power, bypasses HTTP or other protocols (usually implemented as user space applications) and reduces the overall overhead of the communication. The disadvantages are also evident; the TCP functionality is reduced compared to the offering of more sophisticated protocols. All the additional properties that the communication needs (for example authentication, security, compression) have to be implemented directly at the SOAP level. While this might still be a problem there is plenty of work going on that address these aspects.

[23] presents results comparing SOAP over TCP binding with SOAP over HTTP and SOAP over UDP. Their results show that transmission overhead decreases for simpler protocols and that it is most relevant for small sized request and response messages. The increase of size due to the HTTP headers is not significant, but the effects become more visible in WLAN environments. The recommendation is to use TCP when reliable data transfers are necessary but no additional features (like secure transfers, where HTTPS might be a better match) are required.

2.6 UDP

In oder to deal with the poor performance of the HTTP and TCP bindings under specific conditions, companies like Microsoft, BEA Systems and others have proposed a SOAP over UDP binding [17]. This binding has some advantages over bindings that use TCP: it does not require the establishment of a connection, the resource requirements are lower then on the case of TCP based bindings and by supporting multicast new opportunities for developing push based or publish/subscribe web services appear.

This binding also has some disadvantages:

- the SOAP message can not be bigger then 65536 bytes. This is the maximum payload that a UDP datagram can carry and the SOAP message must be small enough to fit as the payload of a UDP datagram.
- it lacks reliable data transfer, due to the nature of UDP. If required by the application a mechanism providing reliability has to be implemented in the application.

[1] http://ws.apache.org/axis2/

[2] http://msdn.microsoft.com/webservices/webservices/building/wse/default.aspx

This implementation effort may make SOAP bindings that provide better reliability a better solution.

- no congestion control is used. This might negatively impact other applications that share the network, especially if an aggressive retransmission system is used or the application generates large data transfers.

The performance results of SOAP over UDP show clear advantages when compared with SOAP over HTTP or TCP [23] but its drawbacks need to be taken into account when evaluating such an option. This binding is suited for applications that need fast but not reliable exchange of preferably small data payloads.

2.7 Other protocols

The presented protocols are not the only one that could be used by SOAP, some other option are:

- BEEP [26] is a communication framework that offers generic, connection oriented, asynchronous communications . It can be mapped onto other protocols, for example TCP. BEEP also offers a SOAP binding [25].
- SCTP [27] is a reliable protocol that has multiple advantages over TCP. Of interest for a SOAP binding would be the message based nature of SCTP, the multi-streaming as well as the multi-homing capabilities of the protocol.
- DCCP [22] is a new protocol that offers similar functionality and tries to improve UDP. An advantage when compared to the SOAP over UDP binding would be the possibility to use congestion control.

While all these protocols offer interesting options, the major impediment, preventing their usage is the scarce availability of implementations and the compatibility problems that would arise.

3 Quantitative Evaluation of SOA Communication Infrastructure

FMC-QE (Fundamental Modeling Concepts - Quantitative Evaluation) is a new method based on FMC [21] and used to understand the quantitative behavior of a system. FMC-QE is part of ongoing work done within the Communication Systems department presented in lecture notes and at the moment prepared for publication [32].

Traditional methods of quantitative evaluation, like queueing networks, stochastic Petri nets or queuing Petri nets are all based on Markov chain representation of the systems. The biggest challenge in all these techniques is to reduce the state spaces. FMC-QE does not follow the state space based approach for quantitatively analyzing

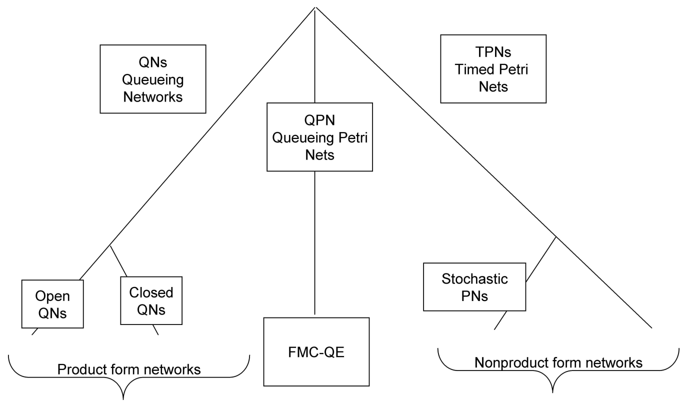


Figure 1: FMC-QE positioning [32]

the system, but it uses hierarchical modeling and fundamental laws. FMC-QE positioning in relation with other modeling techniques is illustrated in figure 1.

FMC-QE make use of two fundamental laws: Little's Law and Forced Traffic Flow Law. Little's Law states that N - the average number of customers in a queueing system is equal to A - the average arrival rate of customers to that system, times R - the average time spent in that system [16]. Little's Law is a "Black Box" law and it can be applied to both aggregated and decomposed components.

$$N_i = A_i \times R_i \tag{1}$$

The second fundamental law used by FMC-QE is the Forced Traffic Flow Law. This law relates the external and internal arrival rates by means of the traffic flow coefficient and is used as the basis for the concept of hierarchical modeling [18].

$$A_{i,internal} = v_{i,internal} \times A_{external} \tag{2}$$

FMC-QE is based on the FMC's 3-dimensional representation of system properties via static, dynamic and value diagrams. One of the strength of FMC-QE is the hierarchical modeling, as a method to reduce the problem of state space explosion, presented in figure 2 along with the clear distinction between the control and operational units in servers, presented in figure 3.

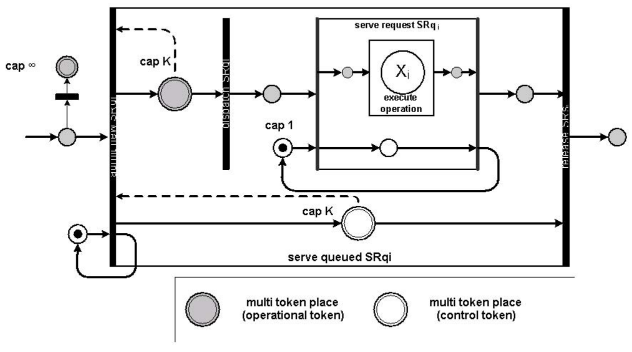


Figure 2: Hierarchical modeling - dynamic structure [32]

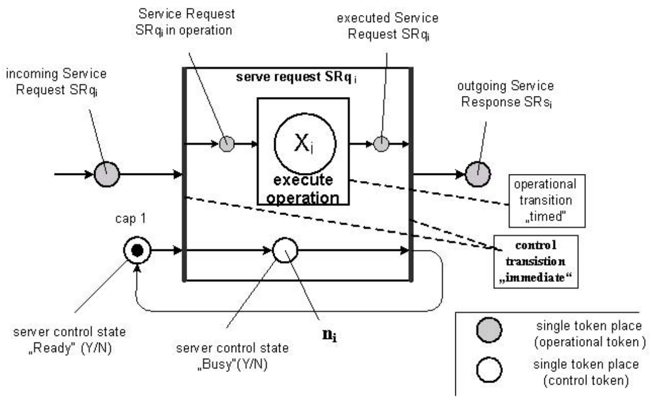

Service Request SRq_i in operation
executed Service Request SRq_i
incoming Service Request SRq_i
serve request SRq_i
outgoing Service Response SRs_i
X_i
execute operation
operational transition „timed"
cap 1
control transistion „immediate"
server control state „Ready" (Y/N)
server control state „Busy"(Y/N)
n_i
single token place (operational token)
single token place (control token)

Figure 3: Separating control and operational units [31]

Although FMC-QE is still being under constant development, we consider that it is a well suited method for modeling the communication stack used by web services. Of special interest are the abstraction capabilities of FMC-QE and the aggregation of lower layer modules via the FMC-QE Tableau.

4 ATrA - Adaptive Transport Architecture

One of the ways of improving service-oriented systems in general and web services in particular is to try and make better use of the available communication infrastructure. A possible way of doing this is by using an adaptive transport architecture that would be able to adapt on the fly to specific network conditions as well as to the current state of the service consumer and service provider. In this section we propose such an architecture, describe the changes required in order to implement it and discuss several scenarios where we expect that this architecture will offer a competitive advantage when used, compared to existing systems.

4.1 Related work

Today's web services should be considered to be transport agnostic but although the web service architecture was developed with this transport independence in mind, most of the web services do run over HTTP [29]. The current W3C Recommendation regarding SOAP based web services [2] states that SOAP messages may be exchanged using a variety of 'underlying' protocols, including other application layer protocols. The SOAP HTTP binding is defined in more details in [2] while a second binding, the SOAP mail binding (that make use of SMTP) is just briefly presented as an alternative option, with details presented in [24].

One way of implementing the SOAP and WSDL recommendation, that specify different transport bindings and encodings, is through an architecture like PEPt [5]. PEPt (presentation, encoding, protocol and transport) has been initially designed as a high level architecture for implementing RPC systems but can be also used for systems based on web services. Presentation encompasses the data types and APIs available to the programmer. Encoding is the representation of those types on the wire. Protocol frames the encoded data to denote the boundaries and intent of the message. Transport moves the encoding and protocol from one location to another. The PEPt architecture enables a single programming model to adaptively change encodings, protocols and transports.

When using such an approach, one aspect of the RPC system may evolve without disturbing the others. In other words, when an alternate encoding, protocol or transport is desired there is no need to create another presentation block. Or, alternatively, a new presentation block can reuse existing protocols, encoding and transports.

The PEPt architecture has been used in the Sun Microsystems implementation of CORBA for Java^{TM}2 [1]. That same implementation has been used to prototype a system that supports RMI-IIOP stubs and ties dynamically switching between IIOP and SOAP/HTTP. The core RPC architecture can serve as the basis for understanding, designing, implementing, maintaining and reusing such RPC systems [6].

A detailed presentation of PEPt architecture on the client and server side together with a performance evaluation are done in [7] and [8]. For performance evaluation a Java implementation of the PEPt architecture has been used for sending 20 instances of a simple Java class while using four different EPTs: Doc-Literal/SOAP/HTTP, RMI-IIOP, ASN.1 Binary XML Encoding/SOAP/HTTP and Java's native RMI. The measurements evaluated the time required for transmission as well as the size of the encoded data for each EPT. The Doc-Literal EPT has the biggest size and it takes longer then any other EPT to be transmitted. The two binary formats are smaller and take less time for transmission, with the native RMI being the smallest and having the fastest transmission times. The binary XML encoding performs well, comparable with the other binary encodings. An extension of the architecture in order to provide high availability for IIOP and SOAP without requiring additional hardware, software nor alterations of the clients is presented in [9].

PEPt enables the possibility to easily change between different combinations of EPTs (encodings, presentation and transports). In this context the 'ease' refers to the amount of effort required to understand each EPT's paradigm and programming model. PEPt can be used in relation with SOA systems even thou the two are at different abstraction levels.

SOA lies at a higher abstraction level and is more concerned about the proper implementation of business logic in the system, orchestration and choreography of the services. As mentioned before, SOA does not really specify the infrastructure that has to be used for communication, the only strict requirement is for the data exchanges to somehow be able to take place.

PEPt is located at a lower abstraction level and could be used by an SOA based system as a very flexible and agile remoting infrastructure. In such an architecture PEPt would be responsible for assuring the proper data exchange between the entities composing the SOA. Providing this separation between the low level document exchange and the high level business logic implementation gives SOA the possibility to integrate with existing infrastructures (ex. CORBA) and services available via such infrastructures. Such a system could also be able to evolve and incorporate new encodings, protocols and transport as these become available and accepted by the community. All this would be possible without any disruption at the higher levels as long as the proper abstraction of the data exchange is preserved.

The usage of PEPt enables a system to adaptively change between encodings, protocols and transports. While this possibility exists, the programmer has to implement all the logic required to detect if different EPT combinations are supported on both ends of the communication system, to identify the moment when a change should be performed and decide what other EPT should be used.

While the possibility to easily switch between different transports is a step in the direction of achieving an adaptive transport infrastructure, such an infrastructure can not be considered without a component capable of deciding what transport should be used and when to switch between different transports.

In [13] and [12] the problem of selecting the best web service from several web services offering the same functionality is approached. Selecting a web service provider for the client is typically a task performed by the client's designer, but their approach focuses on augmenting the clients in order to give them the possibility of dynamically determining the optimal service provider for them. The clients are able to parse, analyze and use their own context information as well as information collected by themselves or other clients regarding the service providers.

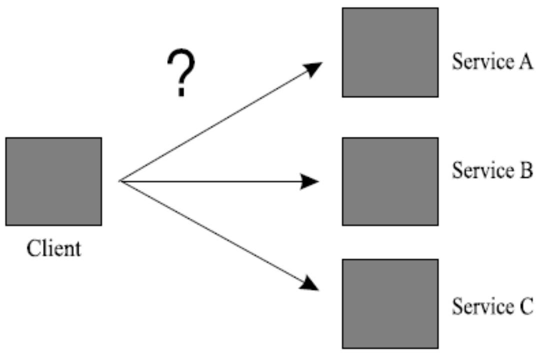


Figure 4: An illustration of the service selection problem

Their solution to the problem of selecting a service between a number of user-specified, syntactically identical web services that provide the same service to the client is to make use of the past experience that other clients have with these services. The approach uses QoS forums where augmented clients share their own QoS measurement regarding different web services. The properties taken into account are generic enough to be applicable to any web service (availability, reliability and execution time) but the system can be extended in a transparent way.

Two approaches have been used for service selection: a rule-based and a simple Bayes reasoner. The rule based approach was used only on the set of data provided by the QoS forum, while the second one also used client context information (processor load, memory usage, number of running processes). Both approaches have been successful, with the first one suffering an initial cost at client startup, while the second one being more flexible.

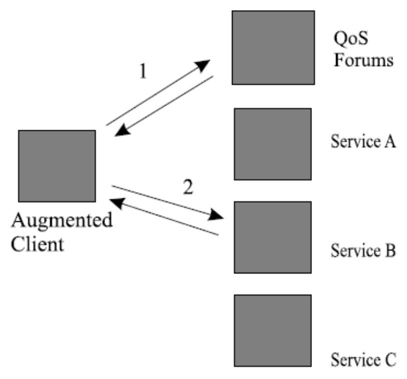


Figure 5: Selecting the best web service through reasoning and the use of QoS Forums

Unlike the problem of web service selection, approached in [13] and [12], an adaptive transport architecture does not have to select between different web services. Such a system will always use one specified web service but will have to decide between using different transport bindings in order to access it. The system could be extended to select between different web service providers, between different transports that could be used in order to access the services as well as taking into account other information, leading to a system that would be completely self adaptive to the changing environment in which such a system exists.

Another difference between the above mentioned system and the adaptive transport architecture that we propose is the fact that ATrA also uses server side augmentation. This has been rejected in [13] because of several reasons, the most important being the fact that servers could try and present an artificially improved set of data to the clients and so trick clients into selecting an inferior service. Due to this, the service providers are treated as black boxes and no information is collected from them.

4.2 Architecture Design

In his most complex form, ATrA or ATrA supporting extensions should be deployed in all three major points of the architecture: service broker, provider and consumer. If the service description along with all the server side data required by ATrA is provided in some other way to the service consumer, ATrA is only required at the consumer and provider as illustrated in 6. As an extreme case, it might be possible to only deploy ATrA at the service consumer, but in this case the capabilities of the system will be limited. Another corner case is the one in which only the server implements ATrA. If that is the case, the server will receive requests using only typical stack, since the client is only capable of invoking the service that way. The service provider will treat these requests just like any other requests whose transport binding has been the result of client side reasoning.

The client side based reasoning (illustrated in figure 7) is based on the following data:

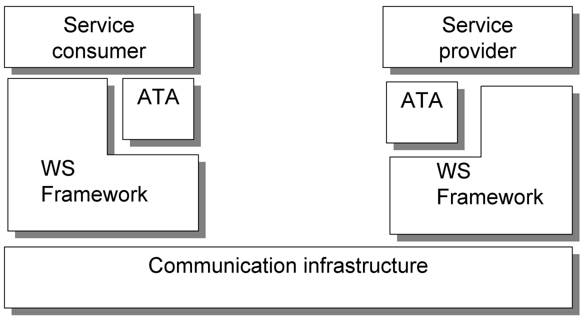


Figure 6: ATrA positioning in the software stack

- Server side data. This information is provided by the server, usually via the service broker. It consists mostly of data describing the server's capabilities, such as supported protocols, supported protocol stacks, preferred protocol stack, preferred protocol settings, server location, etc. Context information could also be made available by the server. Unlike the server's capabilities, which consist mostly of slow changing data, the context information changes very quickly. Due to this data volatility it is not conceivable to disseminate this data via the service broker. One possibility would be to piggyback the data together with a service response. A mechanism can be used to cache the data, selectively publish this data, etc. Another open issue is the security risk presented by providing this kind of data to unknown and possibly malicious clients.
- Client side data.
  - Client capabilities: By this we understand the same as in the case of the server: supported protocols, supported protocol stacks, client location, etc.
  - Context information. Context information is represented by data gathered at the time of the service invocation. This includes, but is not limited to: CPU usage, free memory, percentage of free memory, available bandwidth, number of running processes, etc.
  - Usage profile. This information is provided at the design time or later through user customization. We consider several types of data: requirements, preferences and goals. Requirements strongly affect the possible options of the reasoning engine since they must be enforced. For example if HTTPS usage is a requirement, due to for example security reasons, the transport stack selected must contain the protocol. As another example, REST style web services must use HTTP for transport (as mentioned in 2.2), but we can treat them as a case of services who have a strong restriction regarding the transport infrastructure

The service broker is to be extended so that it can support the richer service descriptions required by ATrA. This can be realized by extending the WSDL (for SOAP web services) in order to support multiple bindings for a web service as well as additional server side data. Similar WSDL extension exist, for example the semantic annotations for WSDL [3].

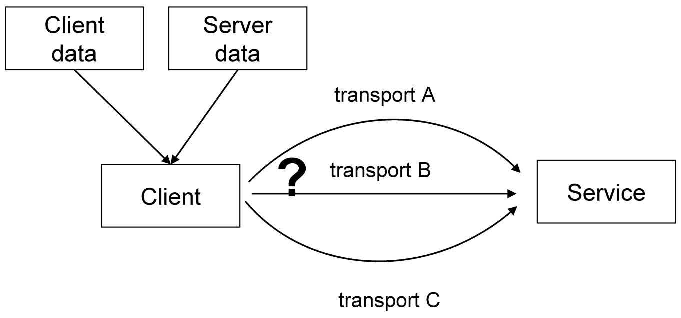


Figure 7: Transport selection based on server and client side data

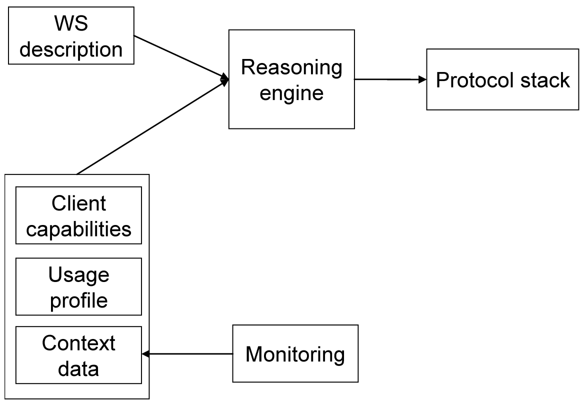


Figure 8: ATrA client architecture

The client and server building blocks are described in figure 8 and 9. When compared to the existing web service implementations some new elements appear:

- Reasoning engine. This is used in order to determine the optimal protocol stack that should be used in order to invoke the service. The reasoning engine uses two main sources of information: the enhanced WSDL description of the service and client side data. The implementation can vary in complexity according to specific needs, from simple rule matching to complex AI algorithms.

- Data set used for reasoning. It is conceivable that not all implementation will use all the available data, some may use only subsets of data in order to improve scalability or system response time. In the case of the server, the protocol stack used by the client for invocation may place additional limitations on the reasoning engine.

- Monitoring engine. This entity is responsible for providing the context information to the reasoning engine. If the framework in use already provides some

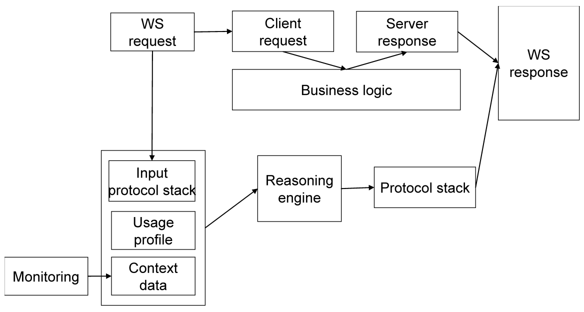


Figure 9: ATrA server architecture

monitoring capabilities, the data can be reused and so the need for another implementation is eliminated.

The proposed architecture enables clients to make context aware decision regarding their selection of the transport stack used for web service invocations. It also enables clients to define and implement highly abstracted profiles to govern their interaction with the service provider (ex. fastest possible invocation, minimum load on server, etc.) The service providers will have the possibly to make services available using multiple transport bindings and to inform clients about ”preferred” protocol stacks. Corporate service providers, that provide services to both the external world as well as to internal clients should also profit from such changes as soon as part of the clients become ATrA enabled. One could envision for ex. a scenario where internal clients could be instructed to use a light-weight protocol during pick load hours to enable the server to better deal with the increased number of requests. Another place where such a platform might prove useful is in improving communication between service providers, especially in the dynamic landscape of service ecosystems.

5 WS invocation over UMTS

The goal of this case study is to investigate the performance of Web Service (WS) invocation over 3G Wide Area Wireless Network (WWAN), such as UMTS, and suggest improvements at the transport and application layer. We would like to know how does 3G WWAN link properties (e.g., long and variable RTT of 400 - 3000 ms, asymmetric links, blackouts) influence WS invocation in different scenarios (synchronous, asynchronous, with or without attachments, different sizes of the request and response, etc.) Is there a functional or performance penalty, and if yes, how to reduce or overcome it at the transport and/or application layer.

It has been previously shown [11] [10] that HTTP downloads seriously underutilize 2.5G wireless links (like GPRS) and several solutions have been proposed at the session, transport and application layers. On the other hand, performance of Web

Services invocation over wireless links has been rarely investigated, in WLAN environment [20] or in emulated GPRS network [28]. Both [20] and [28] focus on possible gains obtained through SOAP message compression but do not even consider the possible functional penalty. To the project participants best knowledge, there has been no published data on the WS invocation performance in UMTS networks. TCP performance measurements almost exclusively deal with either WWW downloads or FTP transfers, while WS performance measurement almost exclusively stop at the SOAP level. Most approaches also deal with downstream communication only (characteristic for WWW download). In WS however, both directions are equally important. This work will therefore observe behavior of the WS invocation starting from (and including) the transport layer.

The goal is to investigate performance of WS invocations over UMTS wireless links. Comparison will be made with WLAN and LAN (wired) links. Different WS payloads will be used (with/without attachments), as well as different communication patterns (synchronous/asynchronous invocation). Underlying protocols will be varied (HTTP, SMTP, JMS over TCP; different versions of TCP will be used (e.g. TCP Westwood), as well as other transport protocols, e.g., SCTP instead of TCP). Different optimizations will be investigated at different layers (e.g., transport layer: persistent connections, modifying TCP packet loss mechanism, improving flow fairness; application layer: proxies, SOAP compression and caching). Only the invocation (binding) part of the WS life cycle will be investigated, assuming that discovery phase already took place. WS invocation performance will be investigated for cases when the WS flow has the whole UMTS link for its usage and for cases when the wireless link is shared with parallel flows of the same type (other WS invocations) and different types (e.g., parallel FTP or HTTP flow). In the preliminary measurements we will concentrate on long and variable RTT, and then depending on the results, include other link properties such as link asymmetry and/or blackouts.

At this time a server dedicated for this experimental investigation has been deployed on the Humboldt University network. The server is running a web service container based on Java Axis 1.4 from Apache on a SuSE Linux operating system. A first set of web services has been developed and deployed on the server. These services offer basic functionality (ex. adding two operands) and support synchronous invocation as well as asynchronous one using SOAP/HTTP/TCP stack. The asynchronous functionality is supported using an extended version of SAIWS (Simple Asynchronous Invocation Framework for Web Services). SAIWS is an add-on to Apache Axis that allows web service client developers to invoke web services asynchronously on top of the HTTP and SOAP protocols (without using messaging protocols) [30].

On the client side a set of corresponding clients that consume the services available from the server has been developed. Service invocation can be done via a script design to automate experiments and perform them in a repetitive fashion , usable for gathering data. The script can invoke one or more web services at a time and do that for a specified number of times. For each invocation the data transferred between the service consumer and the service provider is captured and saved for later processing. The capture is done using tcpdump[3]. For processing the captured data a Java util-

[3]http://www.tcpdump.org/

ity based on Jpcap[4] has been developed. Jpcap is a Java class package that allows Java applications to capture and/or send packets to the network. Jpcap is based on libpcap/winpcap and Raw Socket API and it is supposed to work on any OS on which libpcap/winpcap[5] has been implemented. The Java utility is used to process the capture files, extract the relevant time stamps from the captured packets and save them together with the connection state.

The next step in the project is to acquire and deploy the necessary UMTS equipment in order to be able to start the ”live” experiments. Extending the set of deployed web services along with improvements in the data processing flow are also planned. A move to Axis 2 is also of interest especially due to the wider range of protocol support available in the new Axis version.

6 Conclusions

In this paper we have presented the state of the art in the area of web services communication, discussed the most important protocols used for web service communication as well as some less known and used alternatives. A method suitable for doing performance modeling, evaluation and estimation of the protocol stacks has been presented. Based on these we have proposed an architecture that would enable web service consumers to make informed decisions and select between multiple bindings the one that best suites their needs. The actual status in a project regarding evaluation on the web service invocation has also been presented.

Based on this work we conclude that the communication stack used by web services duplicates much functionality and it has a great potential for improving. We consider very promising the possibility of adapting the communication stack so that it would better suit the specific needs of different classes of clients.

As future work an extensive FMC-QE modeling and evaluation of the Apache Axis framework for web services is intendend. Based on the knowledge gathered during this phase a prototype implementation of ATrA using Axis as the provider for basic web service functionality is planned.

Acknowledgments

Part of the work presented here has been done in collaboration with Bratislav Milic from the Institute for Informatics, Humboldt University Berlin and Nikola Milanovic from the Hasso-Plattner-Institut, University of Potsdam.

References

[1] CORBA Technology and the Java^{TM}2 Platform, Standard Edition.

[4] http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html
[5] http://www.winpcap.org/

[2] SOAP Version 1.2, Jun 2003.

[3] Semantic annotations for WSDL, Sep 2006.

[4] R. J. Bayardo, D. Gruhl, V. Josifovski, and J. Myllymaki. An Evaluation of Binary XML Encoding Optimizations for Fast Stream Based XML Processing. pages 345–354, 2004.

[5] Harold Carr. One-Page PEPt. In *Middleware 2003 Workshop Proceedings*, 2003.

[6] Harold Carr. PEPt - A Minimal RPC Architecture. In *On The Move to Meaningful Internet Systems 2003: OTM 2003Workshops*, volume 2889/2003 of *Lecture Notes in Computer Science*, pages 109–122. Springer-Verlag, November 2003.

[7] Harold Carr. Client-side Encoding, Protocol and Transport Extensibility for Remoting Systems. In *Fifth International Conference on Communications in Computing (CIC)*, June 2004.

[8] Harold Carr. Server-side Encoding, Protocol and Transport Extensibility for Remoting Systems. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 329–334, New York, NY, USA, 2004. ACM Press.

[9] Ken Cavanaugh and Harold Carr. IIOP and SOAP failover in static clusters. In Brian J. d'Auriol and Hamid R. Arabnia, editors, *Communications in Computing*, pages 61–66. CSREA Press, 2005.

[10] R. Chakravorty, S. Katti, J. Crowcroft, and I. Pratt. Flow aggregation for enhanced TCP over wide-area wireless. In *IEEE INFOCOM 2003*, pages 1754–1764, 2003.

[11] Rajiv Chakravorty, Suman Banerjee, Pablo Rodriguez, Julian Chesterfield, and Ian Pratt. Performance optimizations for wireless wide-area networks: comparative study and experimental evaluation. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 159–173, New York, NY, USA, 2004. ACM Press.

[12] J.C. Day. A framework for autonomic web service selection. Master's thesis, University of Saskatchewan, 2005.

[13] Julian Day and Ralph Deters. Selecting the best web service. In *Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative Research*, pages 293–307. IBM Press, 2004.

[14] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, UNIVERSITY OF CALIFORNIA, 2000.

[15] Madhusudhan Govindaraju, Aleksander Slominski, Kenneth Chiu, Pu Liu, Robert van Engelen, and Michael J. Lewis. Toward Characterizing the Performance of SOAP Toolkits. In *GRID '04: Proceedings of the Fifth IEEE/ACM International*

Workshop on Grid Computing (GRID'04), pages 365–372, Washington, DC, USA, 2004. IEEE Computer Society.

[16] Donald Gross and Carl M. Harris. *Fundamentals of Queuing Theory*. John Wiley & Sons, 1998.

[17] M. Gudgin, H. Combs, J. Justice, G. Kakivaya, D. Lindsey, D. Orchard, A. Regnier, J. Schlimmer, S. Simpson, H. Tamura, et al. SOAP over UDP. Technical report, BEA, Lexmark, Microsoft and Ricoh, 2004.

[18] Martin Haas and Werner Zorn. *Methodische Leistunganalyse von Rechenensystemen*. R. Oldenbourg Verlag, 1995.

[19] Jaakko Kangasharju. Mobile XML messaging. Course essay, University of Helsinki, Department of Computer Science, Helsinki, Finland, November 2004.

[20] Jaakko Kangasharju, Sasu Tarkoma, and Kimmo Raatikainen. Comparing SOAP Performance for Various Encodings, Protocols, and Connections. In *Personal Wireless Communications*, pages 397–406, 2003.

[21] Andreas Knpfel, Bernhard Grne, and Peter Tabeling. *Fundamental Modeling Concepts Effective Communication of IT Systems*. John Wiley & Sons, Ltd, 2005.

[22] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol. RFC 4340 (Informational), March 2006.

[23] Kwong Yuen Lai, Thi Khoi Anh Phan, and Zahir Tari. Efficient SOAP Binding for Mobile Web Services. In *LCN '05: Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary*, pages 218–225, Washington, DC, USA, 2005. IEEE Computer Society.

[24] Highland Mary Mountain, Jacek Kopecky, Stuart Williams, Glen Daniels, and Noah Mendelsohn. SOAP Version 1.2 Email Binding, Jun 2002.

[25] E. O'Tuathail and M. Rose. Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP). RFC 3288 (Informational), June 2002.

[26] M. Rose. The Blocks Extensible Exchange Protocol Core. RFC 3080 (Informational), March 2001.

[27] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol. RFC 2960 (Informational), October 2000.

[28] M. Tian, T. Voigt, T. Naumowicz, H. Ritter, and J. Schiller. Performance considerations for mobile web services. *Computer Communications*, 27(11):1097–1105, July 2004.

[29] Werner Vogels. Web services are not distributed objects. *Internet Computing, IEEE*, 7(6):59–66, 2003.

[30] Uwe Zdun, Markus Voelter, and Michael Kircher. Design and implementation of an asynchronous invocation framework for web services. *Web Services - ICWS-Europe 2003*, 2853/2003:64–78, 2003.

[31] Werner Zorn. Distinguishing between Control States and Operational States - a well Proven Paradigm to Cope with the Complexity of Discrete Dynamic Systems.

[32] Werner Zorn. Quantitative Modeling as a Special Abstraction of Systems Modeling.

Asynchronicity and Loose Coupling in Service Oriented Architectures

Nikola Milanovic

nikola.milanovic@hpi.uni-potsdam.de

The paper discusses definition and role of asynchronicity and loose coupling in distributed systems and in service-oriented architectures in particular. Asynchronicity is defined in this context as blocking nature of a client, while loose coupling comprises properties of time- and space-decoupling between client and server. The main focus and goal is to investigate how and to what extent asynchronous behavior and loose-coupling are supported in Web Services Architecture. After examining existing WS frameworks, a sketch of a novel language for describing complex asynchronous and loose-coupled service interactions is presented. The proposed language, called Belinda, is based on the generative communication paradigm.

1 Introduction

The goal of the paper is to revisit definitions and role of asynchronicity and loose coupling, especially in the context of service-oriented systems and architectures. Existing WS frameworks will be investigated and support, benefits and downsides of architecting asynchronous and loosely coupled applications will be identified. Finally, an idea of novel modeling mechanism for service interaction, called Belinda, will be introduced.

The rest of the document is structured as follows: Section 2 identifies important properties related to the definitions of asynchronicity and loose coupling, such as blocking behavior, time and space decoupling, as well as requirements of the asynchronous middleware (e.g., transactional behavior). Section 3 points to the achieving minimum shared understanding among system components as one of the main properties of Service Oriented Architecture and discusses the role of asynchronicity and loose coupling in establishing that goal. Section 4 investigates the means to ensure asynchronous behavior in Web Services Architecture by identifying message exchange patterns and revisiting two styles of service usage (RPC- and message-orientation) and discusses their role in the process of implementing asynchronous and loosely coupled applications. Section 5 introduces the notion of asynchronous middleware and discusses its properties, while Section 6 proposes Belinda language.

2 Asynchronicity and Loose Coupling in Distributed Systems

What does the term asynchronous mean exactly? In the course of time, it has come to indicate *without time*, *without central clock*, *nonblocking* and of course *not synchronous* as well. It should come as no surprise then, that we need to look into the definition of the term synchronous, too. In computer science, synchronous system usually describes either of the two categories: systems (hardware or software) that require access to the common clock or (software) systems where caller of a procedure has to block until a procedure returns operation result [26]. The first category is usually applied to analysis of multiprocessors, network protocols and distributed algorithms. We are interested in the second category, which deals exclusively with program flow and consequently asynchronicity is defined as the behavior where caller does not have to block and wait for a called procedure to finish, but can continue with its own processing and collect the operation result later, using some retrieving mechanism (e.g., callback or polling).

Does this mean that a synchronous distributed system is the one which uses synchronous calls exclusively? Actually not, since the boundaries between synchronous and asynchronous *operations* are rather artificial. Both operation styles are pure programming abstractions and can emulate each other. Understanding the means to perform synchronous and asynchronous operations, however, is the key prerequisite in knowing how to architect synchronous and asynchronous systems. Given the current state-of-the-art in the operating systems, networking and distributing technologies, one can argue that all computer systems are inherently and essentially asynchronous. The level of asynchronicity can be determined not only by the choice of the operation style, but also by investigating use cases or transactional behavior of the whole system or its parts.

Let's take a look at two examples. A good example of a synchronous system behavior is a bank transfer scenario. The task is to transfer an amount of funds between two accounts. It can be solved by calling three operations: authorization, withdrawal and deposit. Caller cannot continue its operation before all three operations complete. Furthermore, if any operation aborts, the entire transfer operation has to be likewise aborted and rolled back. An example of an asynchronous system is a travel reservation scenario. A client invokes operations to make the hotel and flight reservation for a given date, as well as the payment operation. If synchronous approach is adopted, two problems are encountered: long delay and reliability. Checking flight and hotel availability can take a long time and even involve human processing. Therefore, for business cases where transactions can take a long time to complete, it is unacceptable for a client to block and wait. Instead, the client should be free to continue its own processing and be notified when the operations complete, successfully or otherwise. Furthermore, allowing client to continue enables queuing of the requests and performing retry or failover in case some of the operations fail, improving overall fault-tolerance of the system.

We will go into details of procedure- and message-based communication, as well as transaction management in subsequent sections dedicated to Web Services Architecture. Let us now try to explore the issue of loose coupling. The term loose coupling (or decoupling) found its way into colloquial use that we practice today from the field of interprocess communication [25]. It essentially signifies adding isolation between communicating parties (be it processes or clients and servers) in order to achieve the beneficial architectural property. The two properties that are important for our present consideration are decoupling in space and decoupling in time. Decoupling in space denotes anonymous communication between mutually unknown parties, which can be resolved either by introducing a naming, discovery or adaptation system (more on these issues will be given when discussing Web Services). Decoupling in time ensures support for time-disjoint communication, where processes (or clients and servers) with different life cycles, span and lifetime can interact with each other. An example of space decoupled interaction is using an API that enables a client to send a request to the set of Web Services and accept the result of the fastest. On the other hand, time decoupled communication would be a producer/consumer system (described later in more details as design pattern) where in some instances both producer and consumer run simultaneously, but in others they run with certain time offset. Another example would be the one where the requested operation (service) simply does not exist at the time of the request.

There are systems that due to their complexity, scale and distribution cannot be served by the standard, synchronous middleware that enforces Atomicity, Consistency, Isolation, and Durability (ACID) properties using transactions (look back to the fund transfer and travel reservation examples). In such cases, asynchronous middleware is introduced that tries to enforce ACID properties without resorting to the transaction-style synchronization between communicating entities [20]. The challenge is to investigate alternative mechanisms to replace the powerful transactional synchronization, and to come with the new set of primitives equivalent to the concepts of commits and rollbacks. Most of the asynchronous middleware uses messages as basic building blocks, introducing concepts such as message queues, publish/subscribe mechanisms or events as viable alternatives for building consistent and reliable applications. While orientation towards messages will lead us in the direction of SOA, we will return to the concept of asynchronous middleware in Section 5 when we discuss service bus. For now on, we will turn our attention to the service-oriented architecture paradigm and the role asynchronicity and loose coupling play in it.

3 Service Oriented Architecture

The term service-oriented architecture (SOA) emerged in [9] to describe the approach of building loosely coupled distributed systems with minimal shared understanding among system components. The main building blocks in SOA are services. Services are self-describing, open components that support rapid, low-cost development and deployment of distributed applications. The main goal of SOA is transparent, flexible and dynamic interaction of services and their clients over multiple interconnected domains. The benefits of SOA include increased efficiency through task outsourcing and component reuse, easier integration, increased flexibility and agility at business and

IT level, development of composite applications, enabling of multi-vendor application sourcing, and on-demand interconnection with business partners. SOA can be deployed at different levels of granularity: from exposing fine-grained technical functions to coarse-grained business or scientific operations and processes.

The de-facto standard for SOA implementation today is the Web Service Architecture (WSA). It defines SOA as a distributed system in which agents, also known as services, coordinate by message passing [17]:

> A Service-Oriented Architecture (SOA) is a form of distributed systems architecture that is typically characterized by the following properties:
>
> - Logical view: The service is an abstracted, logical view of actual programs, databases, business processes, etc., defined in terms of what it does, typically carrying out a business-level operation.
> - Message orientation: The service is formally defined in terms of the messages exchanged between provider agents and requester agents, and not the properties of the agents themselves. The internal structure of an agent, including features such as its implementation language, process structure and even database structure, are deliberately abstracted away in the SOA: using the SOA discipline one does not and should not need to know how an agent implementing a service is constructed. A key benefit of this concerns so-called legacy systems. By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application that can be ”wrapped” in message handling code that allows it to adhere to the formal service definition.
> - Description orientation: A service is described by machine processable metadata. The description supports the public nature of the SOA: only those details that are exposed to the public and important for the use of the service should be included in the description. The semantics of a service should be documented, either directly or indirectly, by its description.
> - Granularity: Services tend to use a small number of operations with relatively large and complex messages.
> - Network orientation: Services tend to be oriented toward use over a network, though this is not an absolute requirement.
> - Platform neutral: Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the most obvious format that meets this constraint.

Our goal in the remainder of the document will be to examine how to achieve properties identified in the previous section, applied in the WSA environment.

4 Asynchronous Operations in SOA

Let us start with the terminology again: there is no such thing as a synchronous or an asynchronous Web Service. As far as any service is concerned, it will receive an XML message containing the request, process the request (either by mapping it to a remote procedure call or by executing some additional logic upon the message content), and form the reply. The asynchronicity is contained in the way service client interacts with the service. In that sense, we distinguish synchronous and asynchronous invocations of a Web Service. In the simplest case, the same service can be invoked in both ways. Certainly, to handle complex asynchronous patterns, additional logic has to be added to the service but it has no influence on the basic service operation.

An asynchronous client makes a request as part of one transaction and then immediately continues with its own thread of execution. The current client transaction terminates and commits. The service executes in the meantime and generates a response message. This message should be handled by the client within a different thread and in a separate transaction. What is necessary for this to work is a notification and listener mechanism. There must be a correlator (an ID) that is exchanged between the client and the service in order to associate response with request properly.

Previous consideration reflects the API asynchronicity. However, another important factor is the transport that is used to transfer messages between the client and service. Different transport protocols can be used for Web Service communication. Therefore, it's not only the invocation pattern that determines the overall synchronicity. Transport protocols and mechanisms such as HTTPR, JMS or IBM MQSeries Messaging are often described as asynchronous because they offer means for request-response correlation and support push/pull model of message exchange. On the other hand, transport protocols such as HTTP, HTTPS or SMTP are synchronous in a sense that when they are used for asynchronous invocation, application has to provide message correlation or queuing.

Why is the issue of API- and transport- level asynchronicity important? Web Services should be used to offer a wide range of functionalities with different properties, complexity and response times. There is no single best-practice invocation pattern that can be used in all cases. Imagine an example where a protocol with a single transport channel (e.g., HTTP) is used to repeatedly invoke a Web Service that takes a long time to execute in request/response manner. It is to be expected that such a connection will simply time out. On the other hand, imagine a single client that needs to perform simultaneous invocations. Using blocking at the API level would significantly degrade client performance. We will now try to introduce more order by defining blocking and non-blocking API, single and dual transports, as well as all possible combinations of the two.

A blocking API is the simplest invocation pattern where the current thread of client execution blocks and waits until service completes and returns the result. Non-blocking API is the one which uses polling or callback to enable client to continue without suspending its own thread. However, when used with underlying transports that use the single connection to transmit request and response, timeout may occur. Therefore we introduce dual transport where two separate transport connections are used for request and response. The possible combinations of API- and transport-level asynchronicity are shown in Figure 1.

API-level	Transport-level	Capabilities
Blocking	Single	Simple request/response
Non-blocking	Single	Asynchronous callback and polling
Blocking	Dual	Request/response over one-way transport
Non-blocking	Dual	Maximum asynchronous behavior at both levels

Figure 1: Levels of asynchronicity

Different types of interaction between the client and Web Service are called message exchange patterns. WSDL 1.1 defines four basic message exchange patterns [12]:

- one-way
- request/response
- solicit/response
- notification

In one-way invocation, the endpoint receives a message. Request/response describes the exchange where the endpoint receives a message and subsequently sends a correlated response. In the solicit/response pattern, the endpoint sends a message and receives a correlated message from the client. Finally, notification expresses the endpoint which sends a message. Based on them, the following asynchronous message exchange patterns can be formulated [3]:

- one-way with notification
- request/reply
- request/reply with polling
- request/reply with posting

Let's examine each of them in turn (Figure 2). One way with notification uses two messages for request and response which are defined as two separate WSDL operations. The request is one way operation, and response is notification operation. The client should provide correlation ID as well as reply-to address. Each message (one-way and notification) is sent as a separate transmission at the transport level. In asynchronous request/reply, one request/reply operation is defined, but with two messages sent on two separate transport connections. Again, it is the task of the client to provide ID and reply-to address. Both patterns do not include application level acknowledgement, so it would be a good idea to use them with a reliable transport protocol. In the

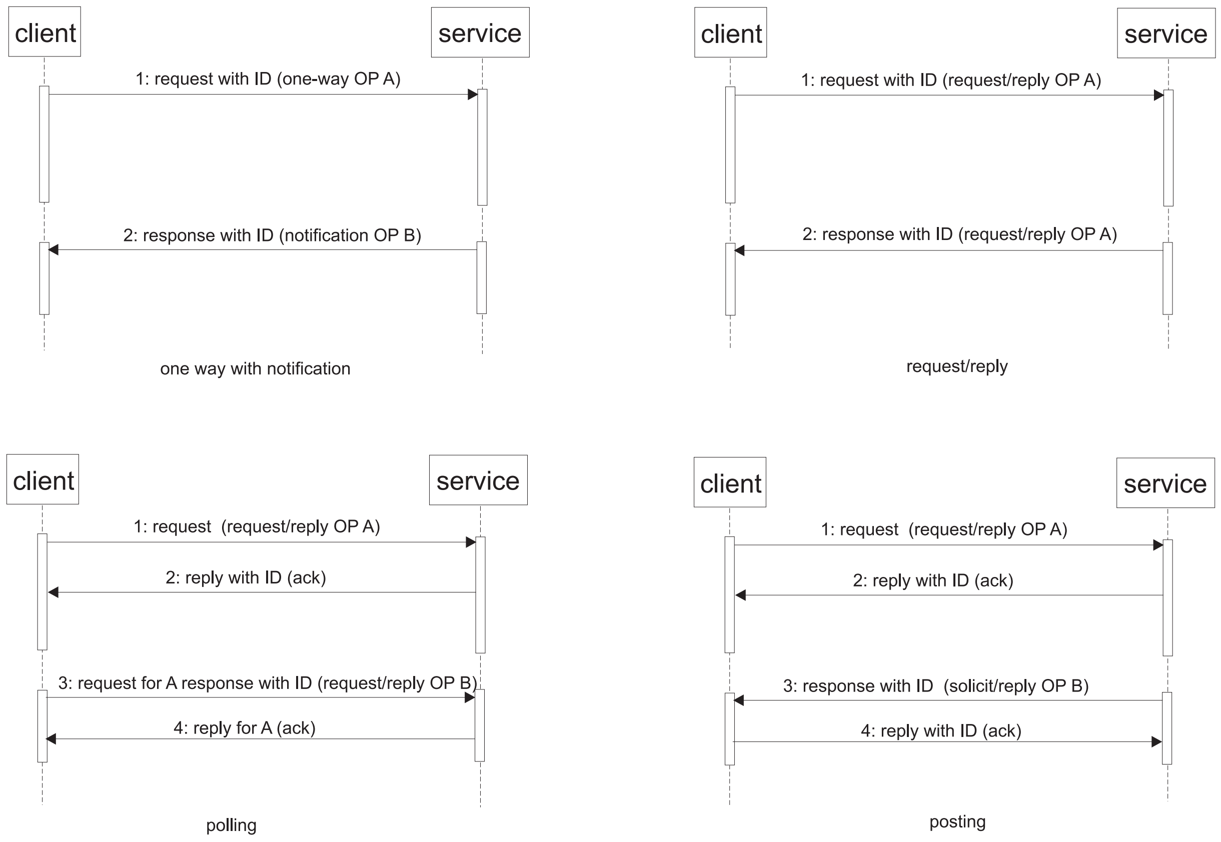

client
service
1: request with ID (one-way OP A)
2: response with ID (notification OP B)
one way with notification
client
service
1: request with ID (request/reply OP A)
2: response with ID (request/reply OP A)
request/reply
client
service
1: request (request/reply OP A)
2: reply with ID (ack)
3: request for A response with ID (request/reply OP B)
4: reply for A (ack)
polling
client
service
1: request (request/reply OP A)
2: reply with ID (ack)
3: response with ID (solicit/reply OP B)
4: reply with ID (ack)
posting

Figure 2: Asynchronous MEPs

polling pattern, four messages are defined inside two WSDL operations. Initial request is request/reply, where the reply part is used for acknowledgment. The service response is retrieved within the second request/reply, where the reply part is interpreted as the operation result. Each pair is implemented on a separate transport connection, but within each connection, operations are synchronous. In this pattern, both sides can generate correlation ID. Finally, posting pattern uses four messages within two WSDL operations, where initial request is modeled as request/reply, and subsequent response is modeled using solicit/reply. The first message in the solicit reply carries the response, while the second one is acknowledgment from the client. Again, request/reply and solicit/response are sent within separate transport connections.

Before proceeding to the implementation issues, we will first clear one problem that frequently causes confusion, namely, SOAP encodings and their relation to the RPC- and document-based applications, as well as the impact they have on the synchronicity. SOAP encoding defines serialization of data inside a SOAP message. Unfortunately, the names chosen to denote different styles of encoding are RPC/literal, RPC/encoded, document/literal and document/literal wrapped. These terms could imply that RPC style should be used for RPC programming models, which are synchronous, and that the document style should be used for document or messaging programming models, which are asynchronous. This is simply not true. The SOAP encoding style has nothing to do with the programming model, despite the names. The SOAP encoding merely dictates how to translate WSDL binding to SOAP message and nothing more. It is possible to use any style with any programming model. The true difference between the RPC and document programming model in SOA lies elsewhere. RPC model means that remote procedure invocation is wrapped in SOAP message. It can be both synchronous or asynchronous. On the service side, SOAP message is decoded and translated into back-end object method invocation. Document programming model is based on the exchange of XML documents without being constrained to execution of predefined methods on the server side. SOAP message does not map directly to a remote method, but rather requires additional processing in order to interpret it, according to a predefined schema. Again, this interaction can be either synchronous or asynchronous, that is, the client sending a message can block and wait until it receives response message or can opt to continue with its own thread of execution.

What are our options for implementation of asynchronous client-service communication in WSA? The fact is that the IDEs and other Web service tooling currently available to automate the generation of the client-side proxies typically only support the synchronous request/response model. This effectively means that developers are discouraged from using asynchronous and document-based communication. Consequence of this are evolution problems. We will now investigate some of the options that are available for implementing asynchronous service invocations, such as Java API for XML Web Services (JAX-WS), WS-Addressing specification, Web Service Invocation Framework, WSDL 2.0 and Axis 2 framework.

The common way of developing Java-based Web Services and clients is to use JAX-WS, which is a continuation of Java API for XML-based RPC (JAX-RPC), within a certain implementation, such as Apache Axis or Java Web Services Developer Pack (JWSDP). JAX-WS offers poor support for asynchronous invocations. Only one asynchronous invocation method is supported, `invokeOneWay`, which supposedly emulates asynchronous one-way invocation with notification. However, even that emulation is not perfect, as the server acknowledges request receipt only after it executes the entire method body, by sending HTTP 200 [1], which is the consequence of using HTTP as the underlying protocol. Effectively, the client still blocks and waits. An option is to spawn a concurrent worker thread on the server side, which would enable client to receive acknowledgement immediately and continue processing.

Extensions to this model have been proposed which develop additional asynchronous message exchange patterns. For example [32] introduces the following patterns based on the JAX-RPC: fire and forget, sync with server, poll object and result callback. Fire and forget is a message exchange pattern where client sends the request and immediately detaches from the service, without waiting for the acknowledgement of receipt or the return value. Sync with server performs similarly, but waits for the receipt acknowledgement from the service. Poll object enables client to periodically poll the service for the results, while result callback enables service to contact the client once the response is ready. The four patterns have been implemented and can be used within Apache Axis framework.

[1]HTTP status code 200, the request has succeeded; the information returned is dependent on the method used.

Departing from JAX-WS, the WS-Addressing specification [8] offers a standard for incorporating addressing information into Web Services messages with the purpose of providing uniform addressing method for SOAP messages that travel over synchronous and asynchronous transports. The interesting part in our context is that it offers addressing options to support building various message patterns. The problem that WS-Addressing tries to address is that WSA and SOAP provide no standard way to specify where a message is going or how to return a response. It has been the task of the transport layer (e.g., URI of the HTTP request is message destination). On the other hand, when a SOAP request is sent asynchronously (e.g., over JMS), response destination may be in the message header, body or left up to the service implementation. The goal of WS-Addressing is to provide capabilities for message routing or directing to a third party. In order to do so, delivery, reply-to and fault handler addressing information is incorporated into a SOAP envelope. WS-Addressing also defines a standard for including service-specific attributes within an address for use in routing the message to a service or for use by the destination service itself (useful for stateful services). WS-Addressing introduces endpoint references and message addressing properties. Endpoint reference is a model for describing destination at which service can be accessed, while addressing properties provide a context for the destination information. When a service receives a message addressed using WS-Addressing, it will also include WS-Addressing headers in the reply message. If a client is sending multiple Web Services requests and receiving asynchronous responses WS-Addressing provides a standard way to associate replies with their corresponding requests. The value of WS-Addressing is most obvious in the asynchronous environment where it offers a consistent addressing model which simplifies integration issues and helps implementing applications where requests are routed to one of several related services using endpoint references.

Returning to the Java world, Web Service Invocation Framework [13] offers a client API for invoking Web Services asynchronously using a local proxy. It offers the capability to use different transports based on the context of the invocation. WSIF supports asynchronous request-response model, where the response is handled in a different thread of execution from the originating request. To support this, the requestor registers a callback object or a handler, that is invoked when the response is received. WSIF also provides correlation and listener services.

The upcoming WSDL 2.0 specification [11] offers a new set of basic message exchange patterns: in-only (single input message), in-out (input message followed by output message), request-response (like in-out but both messages travel on the same channel), in-multi-out (input message followed by one or more output messages), out-only (single output message), out-in (output message followed by input message), out-multi-in (output message followed by multiple input messages) and multicast-solicit-response (output message followed by one or two input messages). Finally, we should mention the new Axis 2 Framework [1], bringing discontinuation with JAX-RPC and introducing AXIOM object model instead. It is very interesting that Axis 2 supports asynchronous service invocation using non-blocking clients and dual transports, as well as all new message exchange patterns of WSDL 2.0. Also, senders and listeners for SOAP over SMTP, FTP and JMS are provided up front.

5 Loose Coupling in SOA

Although SOA advocates loosely-coupled architecture, the primary modus operandi of Web Services is the one based on synchronous request-reply paradigm. However, contrary to the popular belief, Web Services are not (or should not be) just remote procedure calls (RPC) for the Internet. Alternative approaches are neither well documented nor supported in the majority of Web Service frameworks available today. Apart from that, an approach is needed that solves the problem not only within an enterprise, where similar application platforms, native transport and interfaces are used. Today, designers are encouraged to adopt one of the following approaches for basic client-server Web Service invocation: using automatically generated proxy classes or custom SOAP message processing.

In the first approach, a proxy class is generated automatically at design time in the client language based on service's WSDL, prior to client compilation. Proxy class is then compiled together with the client and Web methods are invoked on the proxy object. The benefit of this approach is simplicity: client has the illusion of working with the local object and all issues of SOAP and remote invocation are abstracted from it. The downside is flexibility: client is strongly-coupled with the target Web Service. The client must know the location and relevant names of its partner service in design time in order to generate the proxy class by compiling target WSDL. Target service cannot be changed at runtime without recompiling the client. A client cannot react to the WSDL change at runtime, without being recompiled again. Finally, the problem of serialization/deserialization of SOAP/WSDL types into native language types is neither trivial nor presently solved satisfactorily [21]. Although simple, this approach is exactly the opposite of what Web Service architecture should be: flexible and dynamic, i.e. providing an environment where binding between clients and services is performed on-demand, at runtime. Any serious adaptation in the proxy class approach is not possible without client recompilation.

The second approach expects that a client will generate SOAP requests and parse SOAP response messages. Although theoretically this enables decoupling, such an approach is not practical in design and development of enterprise-grade software systems. The complexity of WSDL files and different flavors of SOAP/WSDL bindings (rpc/encoded, rpc/literal, document/encoded, document/literal, document/literal wrapped) makes it unreasonable to expect that program logic can parse WSDL, create adequate SOAP message and parse response (at least not without unreasonable effort invested). Frameworks exist that help clients in this task but without significant or noticeable success/acceptance and support for true decoupling (anonymity). It is also somewhat difficult to work at the SOAP level unassisted, making this approach unpopular. Several approaches are situated between these two principal extremes (e.g., Dispatcher or Dynamic Invocation Interface in JAX-WS framework) that enable dynamic (runtime) configuration and binding between client and server, but still rely on the built-in serialization/deserialization mechanisms and partially on the use of the proxy classes. Although a step in the right direction, these solutions are not complete.

Finally, whichever approach is chosen, one important issue remains: evolution. By encouraging developers to use Web Services as Internet-enabled RPC extension, potentially rich Web Service design model is downgraded to closed-world RPC where all of the users are known in advance, a data model is shared and all needs can be communicated directly to everyone. Evolution is relatively easy in such an environment: a notification is sent that API is going to change to all relevant parties. If a new system is introduced, point-to-point integration is performed. In the Web Services world, such assumptions are just not realistic: the user base is simply too large. The problems begin when new version of the service is deployed, which is a well-known issue in RPC-based systems. Updates, evolution and versioning of clients and services have to be synchronized and coordinated in such environment, otherwise application will face serious problems. Evolution has to be performed on both sides (client and Web Service) with the minimal shared understanding. RPC-like communication is poorly suited for this scenario, but in spite of that, most developers (encouraged by available tools, or rather discouraged by the absence of support for alternative invocation and communication means) use Web Services in the strong-coupled RPC context, thus making independent evolution difficult to achieve. Web Services architecture should decouple clients and services, but since static invocation (using pre-generated stubs) is the preferred method of issuing remote procedure-based calls, this architectural benefit is lost due to implementation-time assumption of a static service description (static WSDL). Once WSDL document changes, a new set of stubs has to be generated and client application rewritten, recompiled and redeployed. The core of Web Services design should be centered on programming without assumptions paradigm instead: nothing is assumed in advance, everything is discovered on-demand. Independent evolution is necessary for Web Service architecture to be able to scale to the Internet. Therefore, instead of using underlying (software) components as boundaries, data objects (parameters, messages) should be used. In the remainder of this section, we will investigate several options for achieving loose coupling, such as using design patterns, alternative transports and message oriented middleware.

5.1 Design Patterns

Design pattern is a description of the core of an engineering problem that occurs repeatedly in practice, and description of the solution to that problem, such that it is reusable in different contexts. Design patterns were first described in [4] and applied to civil engineering. They were introduced to object-oriented software engineering in [15]. The benefits of design patterns are that they provide high-level language for describing design issues and that combinations of design patterns lead to development of reusable architectures.

Many design patterns have been identified for object-oriented software systems, and some of them have become standard design elements (e.g., Observer, Façade, Command) or have been incorporated in programming languages (e.g., Factory Method). With the introduction of service-oriented computing, it has been noted [23] that usability of ”new” patterns is approaching zero. We are thinking at the new level of granularity when developing service-oriented applications: instead of the object/class, we work at the subsystem/application level. Surprisingly little research is being done in the area of developing methodologies of ”good” service-oriented engineering, with the notable exception of [24] and [5, 6], where several practical Web service design patterns are explained. The design patterns proposed in this section should be considered complementary to them, as well as to SOA Blueprints initiative [30], as ”best-effort” solution for enforcing decoupled design [28]. For the more detailed discussion on the proposed patterns, refer to [22].

There are two main differences between object-oriented and service-oriented design patterns: 1) different levels of design granularity and abstractions; 2) service-oriented design is not inherently client-server, but loosely coupled and dynamic: clients can choose among many servers (services). The goal of the proposed patterns is to identify best practice solutions that are specific to loosely coupled service-oriented design. The following patterns are described: proxy, facade, security, load balancer, logger, dynamic input, producer/consumer and publish subscribe. The patterns are summarized in Figure 3.

The easiest way to access a Web Service is directly, using specific API on the client side that connects to the WSDL interface of a target service. However, this method creates strong coupling between client and called service, and makes reuse difficult, since the same calling code has to be repeated. A solution is to use a service proxy pattern that decouples target service from the client by using surrogate (proxy) service instead of a target service. A proxy pattern should not be confused with the proxy class created by compiling WSDL, a feature offered by most Web Service development frameworks. For a proxy pattern, it is irrelevant how actual service invocation is performed (using a proxy class or generating SOAP messages directly). Single proxy pattern is used to access single Web Service indirectly. The task of a proxy service is to read input parameters, invoke target Web Service and receive results. Therefore, communication with the target service is implemented only once, inside proxy. This facilitates reuse and it is also possible to change the interface of the back-end service without notifying the clients, as it is enough to update only proxy accordingly. The main benefit of using a proxy pattern is that it enforces loose coupling between client and called service. Also, this extra layer of indirection can be used for logging or load balancing, as will be shown later. A service can have more than one proxy. In case of multiple proxies, they are used to convert (transform) interface of the target service according to the expectations of different clients. The alternative name for this pattern is transformer. Transformer is used to enforce understanding on semantic meaning of parameters, or to help connecting services developed with different back-end technologies. Further expansion of the proxy pattern is decoupling of communication protocol and business interface. A channel service is introduced that deals with communication protocol issues while proxy service deals with parameters and business logic. That way communication protocol can be changed without changing either client or service proxy. Multiple proxies can share a single channel, or choose among several available ones.

While proxy pattern facilitates access to a single Web Service, façade pattern performs the same for compositions of services. The problem that façade pattern solves is how to access a composition of Web Services. Contrary to the OO-Façade, multiple network calls are not a problem when invoking Web Service composition, since composite request is sent to the server (e.g., BPEL server) which manages network calls. However, the problem is the coupling between the client and called services.

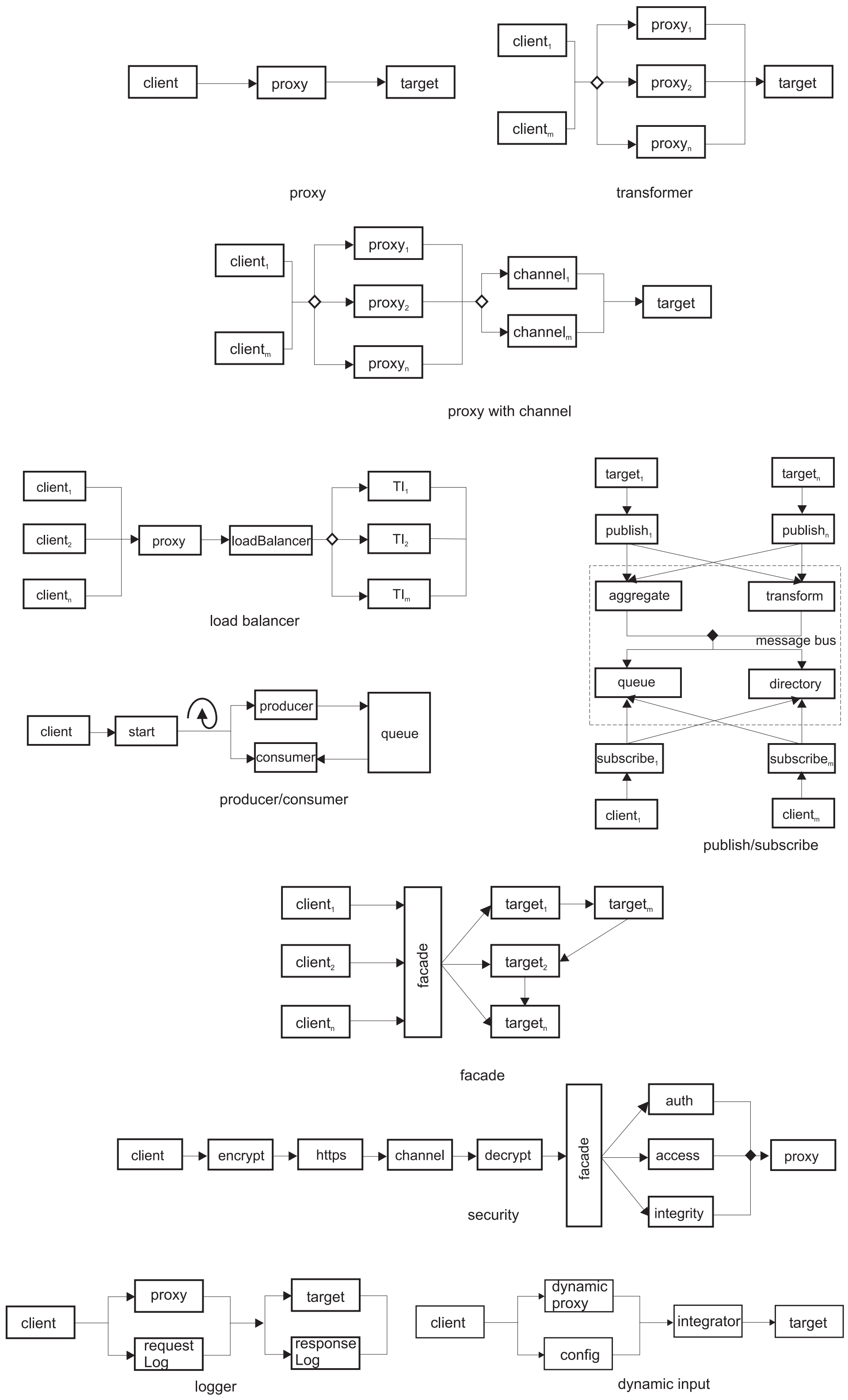


Figure 3: Service design patterns

Although mediated by the composition server, the client has to have intimate knowledge of all services involved. This makes replacement of services in composition difficult. Reusability is not that problematic, as composition can be stored in a directory and then re-invoked when necessary. Finally, in this case it is up to the client and/or composition server to specify non-functional behavior of the constituent services (e.g., transactions) which is a weak design point that can introduce potential inconsistencies. The idea is to represent fine grained operations offered by partner Web Services in a composition through a single Façade service, which offers a coarse grained composite operation to the client. Façade and target services are strongly coupled, but that can be eliminated by applying proxy pattern for each target service. This is the question of the composition complexity, as combining proxy and façade patterns can prove to be an overkill for the application, depending on its size and requirements. The benefit of applying façade pattern is in decoupling client from the partner services comprising the composition. Also it is more natural that the provider of one or more partner services specifies non-functional behavior (e.g., transactions, security, timeliness) instead of the client or the composition server.

There are two ways security requirements can be addressed. The first is to exploit native security capabilities of partner services that build the application. Although easier, it can result in incorrect behavior in case when partner services do not support relevant security properties themselves. The second option is to 'reinforce' an invocation with dedicated services, creating a security pattern or wrapper for the entire operation. Transport protection secures the channel transmitting data from client to facade/ proxy. It encrypts the connection between two services, thus protecting the channel. SSL and HTTPS can be used for this purpose. While in transit, all data is secure. However, data itself is not encrypted, meaning that at the endpoint information is easily read. Therefore, all data is also encrypted before being sent to the channel and decrypted at the receiving end. Finally, after being received by the facade and before being forwarded to proxy, authorization, access control and message integrity verification is performed. It is up to the architect to decide which of these security services should be used in a particular use case, since they introduce significant overhead.

Web Service messages can be very complex, comprising many input/output parameters. It is in accordance with basic postulates of service-oriented computing, which recommend using small number of operations with relatively large and complex messages. Those messages are often not known until runtime, that is, they are constructed dynamically. Data necessary for message construction can come from an XML file, a relational database or from a human user input. Many Web Services also operate with persistent resources which have to be identified (e.g., relational database, table name and primary key). This data is also not known during design time. In such cases it is convenient to remove logic for dynamic message construction outside of the client, into a combination of service proxy and configuration manager. Dynamic proxy decouples client from server and configurator consults persistent resource to retrieve necessary data. Both forward their output to integrator which generates the complete request by filling missing information that it receives from the configurator. In order to remove latency required for consulting persistent resource, configurator caches configuration data in memory, accelerating dynamic message construction and lowering overall response time.

Reputation system is the necessary part of a service-oriented architecture. In order for reputation system to be fair and usable, it is convenient to introduce standard design pattern that requires services to log their input and output messages. That way logging logic is removed from the partner services. Apart from being useful for building standardized reputation systems, logging pattern can be used for debugging and testing, which is often neglected. Developers creating service-oriented applications are often in darkness when trying to debug and diagnose application errors. Even access to error logs is a problem since target services execute in different application servers and access to their logs is not always trivial and/or possible. Introducing standard logging pattern helps in the validation and diagnosis of service-oriented applications. Practice shows that it is recommendable to perform logging in a database, since long XML messages result in unusable log files. It is desirable to have related messages grouped together (concurrency issues). If all related messages are stored in a database, a simple cross join retrieves all relevant data.

Frequently a pool of Web Services that perform the same function is available. Instead of invoking them randomly, they can be load balanced using proxy pattern. Inserting a proxy between clients and a pool of target services, and equipping that proxy with a load balancing algorithm optimizes the performance of the whole system. The role of a load balancer is to distribute client requests among available service instances. If a synchronous load balancer is implemented, only a guessing algorithm can be used, since there is no way that load balancer can know for sure which services are available and which are not. This decision has to be taken based upon imperfect historical data. This is a push model, where requests are pushed to the target service instances without their cooperation. With asynchronous load balancer, a pull model can be used. Load balancer can store requests in a queue and target instances can retrieve and process (pull) them once they are free.

The main actors in the publish/subscribe pattern are publishers, subscribers and communication middleware services (message bus). Publishers produce values (computations, measurements) and put them on the message bus. Subscribers opt to receive a selection of available data from the bus. The bus comprises aggregator, transformer, queue and directory. Aggregators perform data merging, transformers perform transformation on raw data (e.g., a FFT), queue stores raw, aggregated or transformed data, while directory collects descriptions of available data. Subscribers consult directory when choosing data to subscribe to.

The producer/consumer pattern comprises producer, consumer and storage services. Producer receives input data, processes it and puts it into the queue. Consumer takes values from the queue, performs its own processing and sends data to the output. They are both executing asynchronously. Producer and consumer model two elements of a business logic. For example, producer can receive requests for bank transfer, preprocess them (authorization, feasibility) and then put a request in a persistent storage (queue) for producer to take and perform the actual transfer and generate the report. This part of the workflow can involve human interaction. The entire process is executing inside a loop controlled by an external service.

5.2 Alternative Communication Mechanisms

Another way to achieve loose coupling is to investigate alternative communication and transport mechanisms, e.g., using Java Message Service (JMS) or JavaSpaces instead of standard SOAP over HTTP approach. Both JMS and JavaSpaces provide a concept that enables provision of loosely-coupled communication in terms of naming, location and time. Both also provide run-time extensibility and binding, time and location independence as well as latency hiding (asynchronous communication). JMS is principally information delivery platform which offers exchange of structured messages via message oriented middleware. JavaSpaces, on the other hand, is information sharing system.

In the context of SOA, the JMS infrastructure can be understood as a component in the service bus [19]. The basic idea is that service calls are managed and scheduled by the JMS. The infrastructure thus provides a layer between the client and the actual services. The goal is exactly to facilitate asynchronous communication and make it more convenient. Users and services need to agree on the message format in advance, in terms of a contract. Although communication is decoupled using topics and queues, it is still necessary for a client to know how to access the type of message or the topic it is interested in. The boundary between client and service thus becomes the format or content type. That means that they are needed to establish a concrete binding.

In order to use JMS, standard client-service architecture needs to be modified somewhat. From the clients perspective, the endpoint it communicates with isn’t standard Web Service but a protocol handler that listens for SOAP messages and passes them into a message queue. With the aid of a listener, the messages are then routed to the correct target (Web service) asynchronously. The motivation for this approach is that while the Web Services technology enables the execution of remote services, it does not provide a robust infrastructure for information/message handling. The advantages of using JMS for message routing and delivery are numerous. First, the data will be not lost if the application fails because it is persisted. Second, if the system is overloaded with requests, it must be able to handle the increased load. And finally, in the case that the application needs to communicate with a back end system, there must be a bridge for efficient and reliable communication. JMS supports these requirements and provides features to couple both systems, standard SOAP Web Service communication with the outer world and JMS messaging within a network to handle service requests. Besides these advantages, the JMS technology has some drawbacks and problems such as overhead, additional complexity and the risk to form a communication bottleneck. The queues or topics need to maintain and control all incoming messages, perform (de-) serialization of messages and schedule outgoing messages.

JavaSpaces is a technology for loosely coupled communication with respect to location, time and reference. It offers scalable storage and exchange mechanism in form of distributed shared memory make it usable for services, which are executed asynchronously and concurrently. As with JMS, JavaSpaces provide a communication middleware component for services to exchange data. This makes architectures feasible where ”workers” perform the computing by fetching any object from the space and doing its processing locally by calling the computing method. The idea of this approach is to use distributed memory as a shared repository of service requests, descriptions and responses with the main goal of achieving time and space decoupling between clients and the respective services (workers). In the JavaSpaces approach, however, the term service is interpreted in a slightly different way then it is the case for SOAP Web Services. The service as an open, self descriptive component that provides a contract between provider and consumer in SOA does not explicitly exist in this architecture. JavaSpaces comprise object-centric data structures while each object can contain service logic. In fact, since the processing is performed by objects with access to the space, the service itself is distributed over an unknown and not controllable number of systems consuming space objects. When a service is distributed as a data structure written to the space, no service level agreement can be verified. However, JavaSpaces can be combined with standard Web Services in the same manner as it was the case for JMS. The Web Service client uses the well known SOAP interface to call a gateway which distributes the service execution in a space (distributed shared memory). The difference is that with JavaSpaces the main purpose is to distribute workload, not to call third-party systems (like JMS clients), although both are possible with appropriate extensions.

5.3 (Enterprise) Service Bus

Enterprise Service Bus (ESB) [10] is at the same time best-practice design pattern, an architecture and a new type of product. ESB constructs SOA by integrating ”accidental architectures” into a decentralized infrastructure called service bus, which is inherently message-based, asynchronous and loosely coupled. In ESB all applications are provided as (business) services and connected via reliable, secure and managed virtual channels. The main consequence is that orchestration, transformation and maintenance can be moved to the bus and processed in a controlled manner. The structure of the ESB is shown in Figure 4.

The main elements of ESB are: message oriented middleware (MOM), service containers, management facility, routing and XML-processing. Arguably, the most important element is MOM. The task of MOM is to support reliable and asynchronous message exchange. In an ESB architecture, all direct (or legacy) communication channels are replaced by virtual channels, managed by MOM. That way, strongly coupled, synchronous, point-to-point interaction (method invocations) are replaced by loosely coupled indirect interaction implemented using message passing. MOM supports point-to-point (1-1) messaging model as well as publish-subscribe (1-many) model. One example of using MOM is JMS in conjunction with SOAP (see previous section). However, having MOM alone is not enough. The service container is the key element that enables all applications and components to access MOM transport services. To do that, it is connected to topics and queues and is able to transform messages into appropriate service invocations. We can think of service container as an application server (e.g., J2EE). ESB endpoint is part of the service container and is situated between internal services on one side and message clients and MOM on the other side. Continuing with J2EE analogy, endpoint can be understood as a servlet. Apart from the core functionalities (message management, configuring and invoking ESB endpoints), container

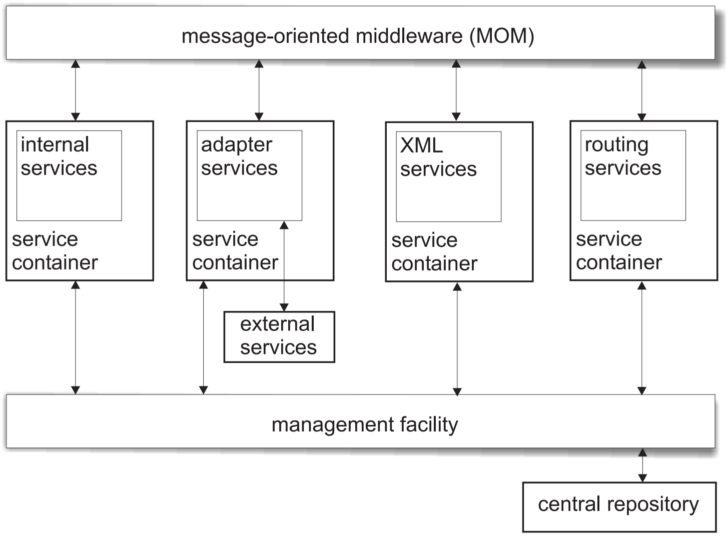


Figure 4: Enterprise Service Bus Architecture

can add any arbitrary function such as transaction management or security.

ESB is based upon a distributed and decentralized structure in which many service containers can be connected via MOM. Therefore, containers, services and MOM need to be configured, monitored and maintained. That is the task of the management facility. The idea is to have decentralized infrastructure which is, however, managed centrally. The management facility comprises central repository and a network of management servers, interfaces and tools. Two other special facilities of the ESB should be mentioned: routing and XML processing. ESB provides three principal means to route messages on the bus: itinerary-based routing, orchestration and content-based routing. Itinerary based routing is used to manage short-living, transient fragments (so called microflows). Each message contains an itinerary that determines its route. The itinerary consists of a list of endpoints that have to be visited and those that have already been visited. On the other hand, orchestration is used to manage long-running transactions using BPEL process definitions. BPEL is provided by ESB as a separate ESB endpoint. Finally, content-based routing employs XML processing facilities which enable validation, transformation and routing of XML messages. Of course, XML facilities can be used for other purposes (e.g., microflows). WS-Policy specifications can be integrated in service containers (e.g., WS-Transactions) or special facilities (e.g., WS-ReliableMessaging).

6 Belinda

In the current research of SOA, usually three major assumptions are made:

- in the service interaction process, discovery has already taken place: consumers have found the adequate providers or composition partners have been identified

- asynchronous, loosely-coupled interaction is trivial to achieve
- the underlying network is ”perfect” (e.g., dependable, efficient and secure)

These assumptions facilitate development of various frameworks for investigating properties of complex service interactions. Both assumptions are, however, also fundamentally false.

As has been argued in the previous sections, current WS frameworks encourage designers to invoke Web Services using synchronous SOAP-RPC. Although it is possible to design asynchronous/loosely-coupled systems that way, the missing support makes this approach unpopular and infeasible. As a direct consequence of this limitation, there is room to propose a more satisfactory model (and middleware) for investigating properties of complex service interactions. Using generative communication paradigm to model invocation, communication and composition, a step away from treating Web Services as an RPC-for-the-Internet can be achieved. In order to mitigate the problems identified with service interactions (discovery, inflexibility, strong-coupling, naming, evolution, fault-tolerance), the idea is to investigate generative communication (Figure 5) as an alternative to classical publish-discover-bind interaction paradigm.

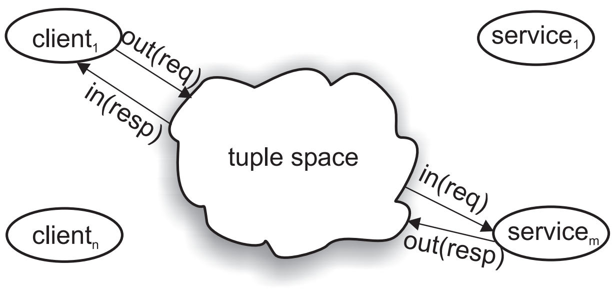


Figure 5: Using generative communication to model service invocation

Generative communication is a distributed programming paradigm that has been proposed as a foundation of the Linda coordination language [16]. The abstract environment called tuple space is the basis of generative communication. Processes involved in a program can generate tuples, store, read or remove them from the tuple space using out, in and read operations. A tuple exists in a tuple space independently of the process that has created it. Generative communication has two distinguishing properties: communication orthogonality and free naming. Communication orthogonality means that the receiver has no prior knowledge about the sender, nor the sender has prior knowledge about the receiver. Direct consequence of this property is space and time uncoupling. Space uncoupling means that any process among many equivalent processes may read a tuple from tuple space (assuming that tuple signature matches positively - that is, assuming that a process is interested in reading such a tuple or that can legally read a tuple). Time decoupling means that a tuple may be written in a tuple space before receiving process is created. Also, a process that has created a tuple may terminate before the tuple has been read by another process. The property of free naming means that variable and operation names are integral parts of tuples. Tuple space ensures that one tuple can be removed by one and only one legal process. Taken together, generative communication enables interaction of space- and time-disjointed processes that can share distributed names and variables with ensured atomicity.

We propose to use a tuple space, shared by clients and services, that mediates between clients' requests and services' responses, creating a language for process-oriented service interaction that we call 'Business-process Execution Linda' (Belinda). The fundamental advantage that Belinda brings is anonymity that results in the loose coupling. Client writes a tuple representing anonymous request into tuple space. One of the available services reads the tuple, performs processing and writes back the tuple representing results. The client can read the tuple and use the results at some point during its execution. This is the simplest kind of asynchronous interaction between a client and a Web Service. Tuples can describe requests, responses, service descriptions and attachments. Tuple space mediates between requests and responses by performing matching of client requests with service capabilities (descriptions). The essence is that the tuple space selects services that can fulfill client's request, without requiring the client to know/discover any particular service in advance. Why is this important? Service-oriented architecture represents a model of a distributed system with minimal shared understanding among system components. By allowing clients not to know their respective partners (Web Services), clients do not have to adapt, which ultimately leads to programming without assumptions. The key is to let the middleware (tuple space) perform necessary adaptations and matching in a standardized way, instead of requiring clients to adapt. In that way, mediation of the broker is elegantly removed and becomes the responsibility of the tuple space. The following benefits of this approach can be identified:

- Anonymity: client does not have to know its target Web Service. This of course doesn't mean that service directory like UDDI is no longer needed. Rather, it will be used by tuple space to perform necessary matching.
- Anonymity leads to loose coupling: service can be changed without any notification to the client. If the new service can still serve the same client, this change will be transparent, otherwise another matching service is located.
- Loose coupling leads to programming without assumptions: flexible invocation, decoupling in space.
- Decoupling in time: easy modeling of synchronous and asynchronous calls, as well as long running transactions.
- Modeling stateful Web Service interactions using tuple space as a context repository.
- Describing service compositions and interaction patterns (synchronous/ asynchronous, publisher/subscriber, producer/consumer...).
- Environment for automatic composition/decomposition/matching analysis.

- Fault-tolerance/responsiveness through automatic and transparent retry, failover and recomposition.
- Quality of service negotiations: services can first offer operations with certain QoS guarantees and clients can opt to accept them or not.

On top of the generative communication model, predefined composition operators will be introduced that enable creation of complex interaction topologies through tuple exchanges. Operators like sequence, parallel, choice, selection or loop will be investigated. The overall Belinda architecture is shown in Figure 6.

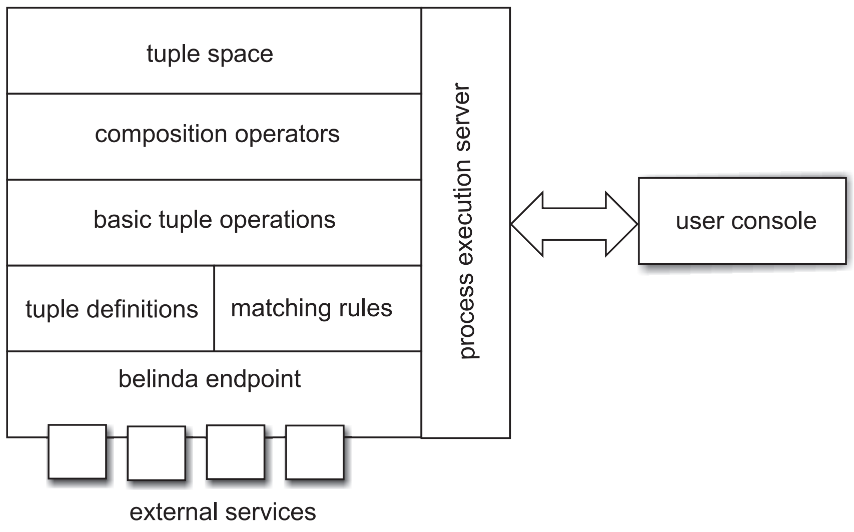


Figure 6: Belinda architecture

Belinda's interface towards external services is called Belinda endpoint and it gives access to tuple space to all interesting parties. Tuple definition comprise requests, responses, attachments and services descriptions. Matching rules determine how tuple signature matching is performed in order to establish communication between two processes (e.g., client and a Web service). Basic tuple operations that will be implemented are in, out, read and eval (tuple creation), with infrastructure guaranteeing unique naming and atomicity. Composition operators are envisioned as high-level language constructs that are defined as 'macros' containing basic tuple space operations and describe complex service interactions (such as synchronous/asynchronous, parallel, sequential, choice, selection or loop execution). Finally, tuple space represents a shared memory in which tuples are stored and read. User communicates with Belinda via process execution server that provides input/output console as well as monitoring tools. An example of asynchronous communication between a client and a Web service can be modeled in the following way:

client:

```
out(method,me,actual_request);
```

```
...
in(me,formal_response)
```

service:

```
in(method,who:name,formal_request) ==>
[procedure;
out(who,actual_response) ]
```

The obvious challenge here is how to model SOAP requests and responses using tuples, how to implement unique method naming, and how to map and match actual and formal parameters (requests and responses). These and other problems will be detailed in the Belinda development plan:

1. Belinda syntax and semantics definition: Belinda syntax will comprise low-level and high-level commands, the former being basic tuple operations (in, out, read and eval) and the latter being composition operators (sequence, parallel, choice, selection, loop). A small command set will enable static program verification. The goal is to define syntax and semantics for the command set and develop grammar that will be used for parsing. The deliverables of this stage should be command set, grammar and parser.

2. Tuple definitions and mappings: The basic units of communication in Belinda are tuples. They represent service requests, responses, descriptions and attachments. Processes (users and services) communicate by writing and reading tuples from a distributed shared memory called tuple space. Algorithms for mapping SOAP requests and responses, service descriptions and attachments will be developed. This will essentially enable a client to issue anonymous service request in form of a tuple, which comprises the mapped SOAP request without defined service endpoint. The process of matching will determine which services can serve the anonymous request. The mapping mechanism between actual and formal parameters will be also developed. The outcome of this phase should be tuple definitions, algorithms for mapping between SOAP/WSDL and tuple notation, and between actual and formal parameters.

3. Tuple space implementation: Tuple space acts as a distributed shared memory and mediates between client and a service, or between composition partners. It has a dual role: supplementing service broker and performing process-based service composition. The key issues are development of unique naming mechanism for processes, methods and parameters and ensuring support for transactional (ACID) properties of basic operations. Basic operations (in, out, read and eval) will be implemented here, as well as following message exchange patterns (MEPs): one-way, request/response, solicit/response and notification. Relevancy and applicability of related approaches (Java Spaces [14], XML spaces [27]) will be investigated. The deliverables expected are definition and implementation of tuple space and basic operations.

4. Definition of matching rules: The fundamental problem of tuple space-based service invocation and interaction is how to define matching rules, namely, which service can accept a client's request, process it and write a response back into tuple space. Purely functional matching based on names/types is not possible here: there are many services accepting the same tuple 'signatures' but performing semantically different operations. The semantic description must therefore be extended not only to services, but also to clients: a client must specify semantics of the request if a tuple space is to provide the best (closest) matching available. Semantic annotations and rules will be defined that will be employed in the matching process. The problem of multiple matches will be also treated. Formalisms and ontologies will be investigated which can potentially describe the semantic issues, such as Frame logic [18], abstract machine notation [2], abstract state machines [7], Web Service Modeling Language [31], Resource Description Framework [29], etc. Deliverables of this phase will be methodologies and algorithms for semantic tuple matching.

5. Composition operators, data flow, process execution: Composition operators are high-level language commands that enable composition of processes (services) into arbitrary topologies using mediation of the tuple space. The basic composition operators that will be defined are sequence, parallel (with and without communication), selection, choice and loop. They will be realized as 'macros' using basic commands (in, out, read, eval). Arity of the composition operators, as well as their properties (commutativity, associativity and distributivity) will be determined. With each operator, a set of data flow rules will be associated, defining how to perform mapping of formal and actual parameters exchanged by the interacting processes. A process execution server (e.g., an BPEL server implementation) is in charge of service composition enactment by scheduling basic Belinda operations with respect to the specified composition. Deliverables expected are composition operators and data flow rules, process execution server implementation/integration.

6. Static Belinda verification: Due to the limited command set, it is expected that static verification of Belinda programs (processes) will be possible to some extent. The intention is to use the specifics of generative communication and its properties to enable following verification forms: type checking, invariant preservation, correct termination and feasibility. This will be done in accordance with the chosen formalism for semantic matching. The role of the static program verification is twofold: preventing incorrect programs to execute and discovering incorrect combinations during the matching process, thus speeding it up and making it more effective. Note that up to this point (due to timing constraints), realization of exception handling mechanism is not planned. We strongly believe that verification and not exception handling should be the preferred way of addressing incorrect behavior of Belinda program. However, this does not mean that in some later stage exception handling mechanism will not be supported. Although a formal exception handling mechanism will not be supported initially, user console and process execution server will monitor process enactment and maintain log file

structure. The outcome of this phase should be a formal method and algorithms for static verification of Belinda programs (processes).

7. User console: User console encompasses standard input/output that serves as a connection to the process execution server. All communication is directed to the process execution server, even in the case of basic client-server invocation. No graphical user interface is planed for now, only terminal window. Belinda programs can be submitted either by typing them in the console, or by loading their description from a file. It will optionally be possible to use the console to include new external services to the tuple space by connecting them to the Belinda endpoint, but this process is not the part of the Belinda syntax and will be performed using pre-defined scripts, activated from within the user console. Deliverable is an input/output mechanism for the user console.

The basic idea behind Belinda, eliminating the service broker using shared distributed memory for achieving space- and time-decoupled communication, presents a radical departure from the well established practice of the service-oriented computing principles. This approach potentially offers a possibility of easy and natural application of message exchange patterns and automated service invocation and composition, which are highly desirable, but at this time not available, features in the area of service-oriented computing.

7 Conclusion

Taking historical perspective into account, SOA is a logical step initiated by the recent developments in the areas of distributed computing and business process modeling, as well as increasing ubiquity of networking technologies. The main goal of SOA is to introduce standard methodologies, languages and protocols for development of distributed applications out of loosely coupled, independent and autonomous software entities. The goal of this document was to identify asynchronous service invocations and loose coupling as enabling architectural properties, offer an overview of available options for realizing them, as well as to sketch a new process-based language Belinda, aimed at modeling asynchronous and decoupled service interactions.

References

[1] Apache Axis 2. *http://ws.apache.org/axis2/index.html*, 2006.

[2] J.R. Abrial. *The B Book*. Cambridge University Press, 1996.

[3] Holt Adams. Asynchronous operations and Web services, Part 2. *http://www-128.ibm.com/developerworks/library/ws-asynch2/index.html*, 2002.

[4] C. Alexander. *A Pattern Language*. Oxford University Press, 1977.

[5] A. Barros and E. Boerger. A Compositional Framework for Service Interaction Patterns and Interaction Flows. In *Proceedings of the Seventh International Conference on Formal Engineering Methods (ICFEM'2005)*, pages 5–35, Manchester, UK, 2005.

[6] A. Barros, M. Dumas, and A. ter Hofstede. Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection. In *Technical Report FIT-TR-2005-02*, Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia, 2005.

[7] E. Boerger and R. Staerk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.

[8] Don Box, Francisco Curbera, and et al. Web Services Addressing (WS-Addressing). *http://www.w3.org/Submission/ws-addressing/*, 2004.

[9] S. Burbeck. The Tao of e-business Services. *Emerging Technologies, IBM Software Group, ftp://www6.software.ibm.com/software/developer/library/ws-tao.pdf*, 2000.

[10] D.A. Chappel. *Enterprise Service Bus*. O'Reilly Media Inc., 2004.

[11] R. Chinnici, J-J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0. *http://www.w3.org/TR/wsdl20/*, 2006.

[12] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. *http://www.w3.org/TR/wsdl*, 2001.

[13] Web Services Invocation Framework. *http://ws.apache.org/wsif/*, 2006.

[14] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley, 1999.

[15] E. Gamma, R. Helm, R. Johnson, and J. Ulissides. *Design Patterns*. Addison-Wesley, 1995.

[16] D. Gelernter. Generative Communication in Linda. *Communications of the ACM*, 7(1), 1985.

[17] W3C Working Group. Web Services Architecture. *http://www.w3.org/TR/ws-arch/*, 2004.

[18] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[19] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. Prentice Hall PTR, 2004.

[20] Doug Lea, Steve Vinoski, and Werner Vogels. Asynchronous Middleware and Services. *IEEE Internet Computing*, 10(1):14–17, 2006.

[21] S. Loughran and E. Smith. Rethinking the Java SOAP Stack. *HP Laboratories Bristol Technical Report, HPL-2005-83*, 2005.

[22] N. Milanovic. Service Engineering Design Patterns. In *Proceedings of the IEEE Symposium on Service-oriented System Engineering*, Shanghai, China, 2006.

[23] G. Prasad, R. Taneja, and V. Todankar. Web and Enterprise Architecture Design Patterns for J2EE. *O'Reilly OnJava, http://www.onjava.com/lpt/a/4161*, 2003.

[24] J.M. Snell. Web services programming tips and tricks: Learn simple, practical Web services design patterns. *www-106.ibm.com/developerworks/library/ws-tip-altdesign1/, /ws-tip-altdesign2/, /ws-tip-altdesign3/, /ws-tip-altdesign4/*, 2005.

[25] Richard Stevens. *UNIX Network Programming, Volume 2, Second Edition: Interprocess Communications*. Prentice Hall, 1999.

[26] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[27] R. Tolksdorf, F. Liebsch, and D. M. Nguyen. XMLSpaces.NET: An Extensible Tuplespace as XML-Middleware. In *Report B 03-08, Free University Berlin*, ftp://ftp.inf.fu-berlin.de/pub/reports/tr-b-03-08.pdf, 2003.

[28] A. van Moorsel. On Best-Effort and Dependability. In *Proceedings of the 2nd International Service Availability Forum (ISAS)*, pages 99–101, Berlin, Germany, 2005.

[29] W3C. Resource Description Framework (RDF). *http://www.w3.org/RDF/*, 2006.

[30] S. Wilkes and J. Harby. *SOA Blueprints Reference Example Requirements Specification*. SOA Center, 2004.

[31] WSML working group. The Web Service Modeling Language WSML. *http://www. wsmo.org/ wsml/wsml-syntax/*, 2006.

[32] Uwe Zdun, Markus Voelter, and Michael Kircher. Design and Implementation of an Asynchronous Invocation Framework for Web Services. In *Web Services - ICWS-Europe*, pages 64–78, 2003.

15	**Application Server Technology**	Peter Tabeling, Rasmus Hofmann, Bernhard Gröne
16	**SAP Application Server Technology**	Peter Tabeling, Bernhard Gröne

ISBN 978-939469-58-2
ISSN 1613-5652