Architecture-Driven Modelling Methodologies

Hannu JAAKKOLA a,1 and Bernhard THALHEIM b,2
a Tampere University of Technology, P.O.Box 300, FI-28101 Pori, Finland
b Christian-Albrechts-University Kiel, Computer Science Institute, 24098 Kiel, Germany

Abstract. Classical software development methodologies take architectural issues as granted or pre-determined. They thus neglect the impact decisions for architecture have within the development process. This omission is applicable as long as we are considering monolithic systems. It cannot however been kept whenever we move to distributed systems. Web information systems pay far more attention to users support and thus require sophisticated layout and playout systems. These systems go beyond what has been known for presentation systems.

We thus discover that architecture plays a major role during systems analysis, design and development. We thus target on building a framework that is based on early architectural decisions or on integration of new solutions into existing architectures. We aim at development of novel approaches to web information systems development that allow a co-evolution of architectures and software systems.

Keywords. architecture-driven development, software development, web, information systems, modelling.

1. Introduction

Typical components of modern information systems are large databases, which are utilized through internet connections. The applications - Web Information systems (WIS) - are usually large and the structure of them is complex covering different types of assets from reusable architectures to COTS components and tailored software elements. The complexity of information systems is increased also the growing demand of interoperability expectations. Larry Boehm - in his conference paper [1] - is using the term “complex systems of systems” in this context. His message is that modern information systems are layered and complex structures based on interoperability between individual systems, products and services.

There is no commonly agreed definition for the notion of a software architecture 3. Some of the notions we found in the literature are too broad, some others are too narrow4. Boehm [2] approaches the topic by analyzing the trends that are worth of knowing in

1Corresponding Author: hannu.jaakkola@tut.fi http://www.pori.tut.fi/~hj
2thalheim@is.informatik.uni-kiel.de http://www.is.informatik.uni-kiel.de/~thalheim
3Compare the large list more than hundred of definitions collected from contributors to http://www.sei.cmu.edu/architecture/start/community.cfm
4Compare http://www.sei.cmu.edu/architecture/start/moderndefs.cfm
adapting the software engineering practices and methods in the current needs. One of his findings points out the importance of architectures. Architectures are means to communicate about software, to set up preconditions to the components and interfaces, to adopt beneficial approaches for strategic reuse in software development, etc. Architecture has three roles:

- to explain: architecture explains the structure of software;
- to guide: architecture guides the designer to follow the predefined commonly accepted rules;
- to enable: architecture provides high level mechanism to implement the requirements set to the product.

In modern software development especially the role of enabling architectures has been growing as the role of reuse as a part of development is increasing.

A similar observation has been made for advanced database system architectures [6,14]. A key observation for database management systems has been that the invariants in database processing determine the architecture of a system. [6] predicted that novel systems such as native XML systems must either use novel architectures or let the user experience the “performance catastrophe”. Business information systems applications that target novel applications, e.g., SOA [15,21], require completely different architectures.

Architecture is a term that must cope with a variety of different aspect reflections and viewpoints. The Quasar model of sd&m [23]) distinguished between the application architecture that reflects the outside or gray-box view of a system, the technical or module construction architecture that separates components or modules for construction and implementation, and the technical infrastructure architecture that considers the embedding of the system into a larger system or into the supporting infrastructure. This separation of concern is similar to different viewpoints in geometry that uses the top view, the profile view, and the ground view. These views are only three views out of a large variety of views.

We use the following definition of the notion architecture: A system architecture represents the conceptual model of a system together with models derived from it that represent (1) different viewpoints defined as views on top of the conceptual model, (2) facets or concerns of the system in dependence on the scope and abstraction level of various stakeholders, (3) restrictions for the deployment of the system and description of the quality warranties of the system, and (4) embeddings into other (software) systems.

We can distinguish five standard views in an architectural framework: (I) The information or data view represents the data that is required by the business to support its activities. This answers the what information is being processed question. (II) The functional business or domain view represents all the business processes and activities that must be supported. This answers the “what business activities are being carried out”. (III) The integration or data-flow view represents the flow of information through the business, where it comes from and where it needs to go. This answers the which business activities require it question. (IV) The deployment or technology view represents the

---

5 The conceptual model includes structural, behavioural and collaboration elements. Systems might be modularised or can also be monolithic. The conceptual model allows us to derive a specification of the system capacity. We may distinguish between standard views and views that support different purposes such as system construction, system componentisation, documentation, communication, analysis, evolution or migration, mastering of system complexity, system embedding, system examination or assessment, etc.
physical configuration and technology components used to deploy the architecture in the operating environment. This answers the where is the information located question. (V)

The *infrastructure* or *embedding* view represents the system as a black- or grey-box and concentrates on the embedding of the system into other systems that are either supporting the system or are using systems services.

Web information systems are typically layered or distributed systems. Layering and distribution results in rather specific data structures and functions that are injected in order to cope with the specific services provided by layers or components. The *CottbusNet* projects used a multi-layer and distributed environment. For instance, the events calendar in city information systems may use a dozen or more different database systems and a view tower. A view tower of such systems must provide advanced search facilities [4]. It uses views that compile a variety of ETL results into a common view for events, an extraction view for presentation of events at a classical website or at other media such as video text canvas or smart phone display, a derived search functionality for these data, and a collection view for a shopping cart of an event shopper. A similar observation can be made for OLTP-OLAP systems [12,13]. OLAP systems are typically built on top of OLTP systems by applying first grouping and aggregation functions and second by integrating data obtained into a data mart presentation.

In projects aiming in developing web information systems [25] we discovered that interactivity required redevelopment and adjustment of functionality and of structuring of supporting database systems. Therefore, the presentation layer of a system “struck through” to the support system and resulted in change of this system. This observation complements the observations such as [6,14,21] and shows that web information systems must be build on a more flexible consideration of architectures.

These observations can be summarized into the *architecture/application impedance mismatch*: Architecture solutions heavily influence the capability of a system and must be considered as an orthogonal dimension during systems development.

*Outline of the Paper*

This paper opens discussion on Architecture-Driven Modelling Methodologies in the connection with large Web Information Systems. The paper has its roots in a joint research project of the co-authors; the project has had connections to other related research activities of the participating organisations, and it is funded by DAAD in Germany and Academy of Finland. This paper provides an overview to the approach and methodology developed in the project. Sections 2 introduces the key concepts of the paper. Sections 3 and 4 cover the bindings of the topic to the state-of-the-art of classical IS methodologies and to the Co-Design approach developed by one of the co-authors [25,19]. Architecture Driven Methodologies are discussed in Section 5. The paper summarises the findings of the project by introducing a four-dimensional or four-facetted model to software development in Section 6.

2. Architecture-Driven Modelling of Web Information Systems

2.1. The Challenges of Modern Web-Based and Web Information Systems

Web information systems (WIS) [3,9,20] augment classical information systems by modern Web technologies. They require at the same time a careful development and sup-
port for the interaction or story spaces beside the classical support for the working space of users. These dimensions complicate the system development process. Usually, WIS are data-intensive applications which are backed by a database. While the development of information systems is seen as a complex process, Web information systems engineering adds additional obstacles to this process because of technical and organizational specifics:

- WIS are open systems from any point of view. For example, the user dimension is a challenge. Although purpose and usage of the system can be formulated in advance, user characteristics cannot be completely predefined. Applications have to be intuitively usable because there cannot be training courses for the users. Non-functional properties of the application like ‘nice looking’ user interfaces are far more important compared with standard business software. WIS-E is not only restricted to enterprises but is also driven by an enthusiastic community fulfilling different goals with different tools.
- WIS are based on Web technologies and standards. Important aspects are only covered by RFCs because of the conception of the Internet. These (quasi-)standards usually reflect the ‘common sense’ only, while important aspects are handled individually.
- Looking at the complete infrastructure, a WIS contains software components with uncontrollable properties like faulty, incomplete, or individualistically implemented Web browsers.
- Base technologies and protocols for the Web were defined more than 10 years ago to fulfill the tasks of the World Wide Web as they had been considered at this time. For example, the HTTP protocol was defined to transfer hypertext documents to enable users to browse the Web. The nature of the Web changed significantly since these days, but there were only minor changes to protocols to keep the Holy Cow of Compatibility alive. Today, HTTP is used as a general purpose transfer protocol which is used as the backbone for complex interactive applications. Shortcomings like statelessness, loose coupling of client and server, or the restrictions of the request-response communication paradigm are covered by proprietary and heavy-weight frameworks on top of HTTP. Therefore, they are not covered by the standard and handled individually by the framework and the browser, e.g., session management. Small errors may cause unwanted or uncontrollable behavior of the whole application or even security risks.

WIS can be considered from two perspectives: the system perspective and the user perspective. These perspectives are tightly related to each other. We consider the presentation system as an integral part of WIS. It satisfies all user requirements. It is based on real life cases. Software engineering has divided properties into functional and non-functional properties, restrictions and pseudo-properties. This separation can be understood as a separation into essential properties and non-essential ones. If we consider the dichotomy of a WIS then this separation leads to a far more natural separation into information system requirements and presentation systems requirements. The system perspective considers properties such as performance, efficiency, maintainability, portability, and other classical functional and non-functional requirements. Typical presentation system requirements are usability, reliability, and requirements oriented to high quality in use, e.g., effectiveness, productivity, safety, privacy, and satisfaction. Safety and security are also considered to be restrictions since they specify undesired behavior of systems.
Pseudo-properties are concerned with technological decisions such as language, middleware, operating system or are imposed by the user environment, the channel to be used, or the variety of client systems.

WIS must provide a sophisticated support for a large variety of users, a large variety of usage stories, and for different (technical) environments. Due to this flexibility the development of WIS differs from the development of information systems by careful elaboration of the application domain, by adaptation to users, stories, environments, etc.

Classical software engineering typically climbs down the system ladder to the implementation layer in order to create a productive system. The usual way in today’s WIS development is a manual approach: human modelling experts interpret the specification to enrich and transform it along the system ladder. This way of developing specifications is error-prone: even if the specification on a certain layer is given in a formal language, the modelling expert as a human being will not interpret it in a formal way. Misinterpretations, misunderstandings, and therefore the loss of already specified system properties is the usual business.

2.2. The Classical Presentation System Development for Web Information Systems

Classical approaches to web information systems are often based on late integration of presentation systems into the WIS information system. This approach is depicted in in Figure 1.

Classically several layers of abstraction are identified. The top layer is called the application domain layer. It is used to describe the system in a general way: What are the intentions? Who are the expected users?

The next lower layer is called the requirements prescription layer, which is used to concretise the ideas gathered on the application domain layer. This means to get a clearer picture of the different kinds of users and their profiles. This may also include the different roles of users and tasks associated with these roles. The major part of this layer, however, deals with the description of the story board. Stories identify possible paths through the system and the information that is requested to enable such paths. So the general purpose of the business layer is to anticipate the behaviour of the system’s users in order to set up the system in a way that supports the users as much as possible.

The central layer is the conceptual layer. Whilst the requirements prescription layer did not pay much attention to technical issues, they come into play on the conceptual layer. The various scenes appearing in the story board have to be analysed and integrated, so that each scene can be supported by a unit combining some site content with some functionality. This will lead to designing abstract media types. The information content of the media types must be combined to design the structure of an underlying database.

The next lower layer is the presentation layer which is devoted to the problem of associating presentation options to the media types. This can be seen as a step towards implementing the system.

Finally, the lowest layer is the implementation layer. All the aspects of the physical implementation have to addressed on this layer. This includes setting up the logical and physical database schemata, the page layout, the realisation of functionality using scripting languages, etc. As far as possible, components on the implementation layer, especially web-pages, should be generated from the description on the higher layers.
This approach has the advantage that the presentation system specification is based on database views. The entire presentation depends on the maturity of the information systems specification. For this reason we may prefer the development according to the methodology depicted in Figure 1 or better in Figure 4.


ARIS (Architecture of Integrated Information Systems) [16] defines a framework with five views (functional, organizational, data, product, controlling) and three layers (conceptual (‘Fachkonzept’), technical (‘DV-Konzept’), and implementation). ARIS was designed as a general architecture for information systems in enterprise environments. Therefore, it is too general to cover directly the specifics of Web information systems and needs to be tailored.

The Rational Unified Process (RUP) [10] is an iterative methodology incorporating different interleaving development phases. RUP is backed by sets of development tools.
RUP is strongly bound to the Unified Modelling Language (UML). Therefore, RUP limits the capabilities of customization. Like ARIS, RUP does not address the specifics of WIS-E. A similar discussion can be made for other general purpose approaches from software engineering.

OOHDM [22] is a methodology which deals with WIS-E specifics. It defines an iterative process with five subsequent activities: requirements gathering, conceptual design, navigational design, abstract interface design, and implementation. OOHDM considers Web Applications to be hypermedia applications. Therefore, it assumes an inherent navigational structure which is derived from the conceptual model of the application domain. This is a valid assumption for data-driven (hypermedia-driven) Web applications but does not fit the requirements for Web information systems with dominating interactive components (e.g., entertainment sites) or process-driven applications. There are several other methodologies similar to OOHDM. Like OOHDM, most of these methodologies agree in an iterative process with a strict top-down ordering of steps in each phase. Surprisingly, most of these methodologies consider the implementation step as an ‘obvious’ one which is done by the way, although specifics of Web applications cause several pitfalls for the unexperienced programmer especially in the implementation step. Knowledge management during the development cycles is usually neglected.

There are several methodologies that cope with personalization of WIS. For example, the HERA methodology [7] provides a model-driven specification framework for personalized WIS supporting automated generation of presentation for different channels, integration and transformation of distributed data and integration of Semantic Web technologies. Although some methodologies provide a solid ground for WIS-E, there is still a need for enhancing the possibilities for specifying the interaction space of the Web information system, especially interaction stories based on the portfolio of personal tasks and goals.

This list of projects is not complete. Most of the project are not supporting conceptual development but provide services for presentation layout or playout. The Yahoo pipes project6 uses mashup services for remixing popular feed types. The Active Record pattern embeds the knowledge of how to interact with the database directly into the class performing the interaction.

4. Co-Design of Web Information Systems

We distinguish a number of facets or views on the application domain. Typical facets to be considered are business procedure and rule facets, intrinsic facets, support technology facets, management and organization facets, script facets, and human behavior. These facets are combined into the following aspects that describe different separate concerns:

- The structural aspect deals with the data which is processed by the system. Schemata are developed which express the characteristics of data such as types, classes, or static integrity constraints.
- The functional aspect considers functions and processes of the application.
- The interactivity aspect describes the handling of the system by the user on the basis of foreseen stories for a number of envisioned actors and is based on media

---

6See: http://pipes.yahoo.com
objects which are used to deliver the content of the database to users or to receive new content.

- The distribution aspect deals with the integration of different parts of the system which are (physically or logically) distributed by the explicit specification of services and exchange frames.

Each aspect provides different modelling languages which focus on specific needs. While higher layers are usually based on specifications in natural language, lower layers facilitate formally given modelling languages. For example, the classical WIS Co-Design approach uses the Higher-Order Entity Relationship Modelling language for modelling structures, transition systems and Abstract State Machines for modelling functionality, Sitelang for the specification of interactivity, and collaboration frames for expressing distribution. Other languages such as UML may be used depending on the skills of modelers and programmers involved in the development process.

A specification of a WIS consists of a specification for each aspect such that the combination of these specifications (the integrated specification) fulfills the given requirements. Integrated specifications are considered on different levels of abstraction (see Figure 2) while associations between specifications on different levels of abstraction reflect the progress of the development process as well as versions and variations of specifications.

Unfortunately, the given aspects are not orthogonal to each other in a mathematical sense. Different combinations of specifications for structure, functionality, interactivity, and distribution can be used to fulfill given requirements while the definition of the ‘best combination’ relies on non-functional parameters which are only partially given in a formal way. Especially the user perspective of a WIS contributes many informal and vague parameters possibly depending on intuition. For example, ordering an article in an online shop may be modelled as a workflow. Alternatively, the same situation may be modelled by storyboards for the dialog flow emphasizing the interactivity part. This principle of designing complex systems is called Co-Design, known from the design process of embedded systems where certain aspects can be realized alternatively in hardware or software (Hardware Software Co-Design). The Co-Design approach for WIS-E developed in the Kiel project group defines the modelling spaces according to this perception.

We can identify two extremes of WIS development. Turnkey development is typically started from scratch in a response to a specific development call. Commercial off-the-shelf development is based on software and infrastructure whose functionality is decided upon by the makers of the software and the infrastructure than by the customers. A number of software engineering models has been proposed in the past: waterfall model, iterative models, rapid prototyping models, etc. The Co-Design approach can be integrated with all these methods.

At the same time, developers need certain flexibility during WIS engineering. Some information may not be available. We need to consider feedback loops for redoing work that has been considered to be complete. All dependencies and assumptions must be explicit in this case.

In [5] we discussed one strategy to early incorporate architectural concerns into website development. The outcome was a methodology with a third development step that aims in the development of a systems architecture before any requirements elicitation is deployed.
Architectural styles provide an abstract description of general characteristics of a solution. The following table list some of the styles.

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-Server</td>
<td>Segregates the system into two applications, where the client makes a service request to the server.</td>
</tr>
<tr>
<td>Component-Based Architecture</td>
<td>Decomposes application design into reusable functional or logical components that are location-transparent and expose well-defined communication interfaces.</td>
</tr>
<tr>
<td>Layered Arch.</td>
<td>Partitions the concerns of the application into stacked groups (layers).</td>
</tr>
<tr>
<td>Message-Bus</td>
<td>A software system that can receive and send messages that are based on a set of known formats, so that systems can communicate with each other without needing to know the actual recipient.</td>
</tr>
<tr>
<td>N-tier / 3-tier</td>
<td>Segregates functionality into separate segments in much the same way as the layered style, but with each segment being a tier located on a physically separate computer.</td>
</tr>
<tr>
<td>Object-Oriented</td>
<td>An architectural style based on division of tasks for an application or system into individual reusable and self-sufficient objects, each containing the data and the behavior relevant to the object.</td>
</tr>
<tr>
<td>Separated Presentation</td>
<td>Separates the logic for managing user interaction from the user interface (UI) view and from the data with which the user works.</td>
</tr>
<tr>
<td>SOA</td>
<td>Refers to Applications that expose and consume functionality as a service using contracts and messages.</td>
</tr>
</tbody>
</table>

Each of these styles has strengths, weaknesses, opportunities, and threats. Strengths and opportunities of certain architectural styles are widely discussed. Weaknesses and threats are discovered after implementing and deploying the decision. For instance, the strengths of SOA (service oriented architecture) are domain alignment, abstraction, reusable com-
ponents, and discoverability. Weaknesses of SOA are acceptance for SOA within the organization, harder aspects of architecture and service modeling, implementation difficulties for a team, methodologies and approaches for implementing SOA, and missing evaluations of various commercial products that purport to help with SOA rollouts. Threats of SOA are the development of a proper architectural plan, the process plan, resource scope, the application of an iterative methodology, the existence of a governance strategy, and the agreement on clear acceptance criteria.

Therefore, a selection of an architecture has a deep impact on the web information system itself and drives the analysis, design and development of such systems. Figures 1 and 4 consider a separation of systems into a presentation system and a support system, i.e. the classical client-server decision. The picture is more complex if we decide to use 3-tier, SOA or other architectures. The structuring and the functionality that are provided by each of the subsystems must be properly designed. Therefore, the architectural style is going to drive the development process.

5. Architecture-Driven and Application-Domain-Ruled Modelling Methodologies

The project we report was aiming in bridging two technologies developed in the research groups at Kiel and Tampere universities. The Tampere team has been concentrating in the past on software development technologies and methodologies. They have been contributing to corresponding standards. The Kiel team has gained deep insight into web information systems development. In the past the two groups have already been collaborating for the development of a web information systems design methodology.

We built a framework that is based on early architectural decisions or on integration of new solutions into existing architectures. We aim in development of novel approaches to web information systems development that allow a co-evolution of architectures and software systems.

WIS development results in a number of implemented features and aspects. These features and aspects are typically well-understood since they are similar to classical software products. One dimension that has often been taken into consideration at the intentional level is the level of detail or granularity of the description. Classical databases schemata are, for instance, schemata at the schema level of detail. This schema level is extended by views within the three-level architecture of database systems. These views are typically based on macro-schemata. Online analytical processing and data warehouse applications brought another level of detail and are based on aggregated data. Content management systems are additionally based on annotations of data sets and on concepts that explain these data sets and provide their foundation. Finally, scientific applications require another schema design since they use sensor data which are compacted and coded. These data must be stored together with the ‘normal’ data.

The architectural component has been neglected for most systems since architecture has been assumed to be canonically given. This non-consideration has led to a number of competing architectures for distributed, main-frame or client-server systems. These architectures can however been considered as elements of the architecture solution space.

Therefore the development space for software systems development can be considered to be three-dimensional. Figure 3 displays this space.

Web information systems development has sharpened the conflicting goals of system development. We must consider at the same time a bundle of different levels of details,
languages and schemata. Systems will not provide all features and aspects to all users. Users will only get those services that are necessary for their work. At the same time, a number of architectural solutions must co-exist.

5.1. Development by Separation of Concern

Our approach concentrates on the separation of concern for development. We shall distinguish the user request diploid within a development:

Application domain modelling aims in meeting the expectations of the user depending on their profile and their work portfolio. Users want to see a system as companion and do not wish to get another additional education before they can use a system.

Architecture modelling proposes a realisation alternative. This architecture is typically either based on already existing solutions or must be combined with the user system.

Separation of concern for development allows to decompose an application into fields of action, thought or influence. All components have an internal structure formed from a set of smaller interlocking components (sub-component) performing well-defined functions within the overall application domain. Separation of concern covers the what, who, when and (if its relevant) the why aspects of the business and allows us to identify ‘owners’ and ‘influencers’ of each significant business activity that we need to consult whenever we want to change any of these aspects. A prescriptive (i.e., principles driven) separation is easier to justify to business stakeholders when proposals are put forward to restructure a business activity to improve overall efficiency.

Functional business areas have a high influence on a system. They are identifiable vertical business areas such as finance, sales & marketing, human resources or product manufacturing; and in other cases, they are cross-functional “horizontal” areas such as customer service or business intelligence. Therefore, the business areas already govern the architecture of a system. The establishment of an “ownership” of an information flow assigns the owner to be responsible for making the data available to other business areas as and when those business areas require it. “Influencers” of an information flow need to be consulted when any changes are proposed to ensure that they can comply with the
change. Coherence boundaries are the points at which different functional business areas have to communicate with the outside world in a consistent and grammatically structured language.

This request diploid is mapped then to different systems and can be separated as shown in Figure 1. We typically distinguish between the

**user system** e.g. consisting of the

- **presentation system** and possibly of
- **supporting systems** and the

**computer system** which uses a certain architecture, platform and leads to an implementation.

Based on the abstraction layer model in Figure 2 we may distinguish different realisations of systems:

**Information-systems-driven development** is based on late integration of the presentation and user system. Presentation systems are either developed after the conceptualisation has been finished (this leads to the typical ladder in Figure 1) or are started after the implementation has been developed. In this case we distinguish the following phases:

1. application domain description;
2. requirements elicitation, acquisition, and compilation prescription;
3. business user layer;
4. conceptual layer;
5. implementation layer.

**Web information systems** use more flexible architectures. Their development is intentionally often already based on development methodologies presented in Figure 4. So far, no systematic development of an methodology beside the methodology developed in our collaboration has been made. We typically may distinguish the following phases:

1. application domain description;
2. requirements elicitation, acquisition, and compilation prescription;
3. conceptual systems layer;
4. presentation systems layer;
5. implementation layer.

Additionally we may also consider the deployment, maintenance, ... etc. layers. We restricted our project to the layers discussed above.

### 5.2. Abstraction Layering During Systems Development

Our approach allows to integrate architecture development with architecture development. Top-down development of systems seems to be the most appropriate whenever a system is developed from scratch or a system is extended. For this reason, we may differentiate among three layers: the systems description and prescription layer, the conceptual specification layer, and the systems layer. These layers may be extended by the strategic layer that describes the general intention of the system, by the business user layer that describes how business users will see the system and by the logical layer that relates the conceptual layer to the systems layer by using the systems languages for pro-
gramming and specification. Figure 4 relates the three main layers of systems development.

The system ladder distinguishes at least between the following refinement layers: description / prescription, specification, and implementation. The refinement layers allow to concentrate on different aspects of concern. At the same time, refinement is based on refinement decisions which should be explicitly recorded. The implementation is the basis for the usage. The dichotomy distinguishes between the user world and the system world. They are related to each other through user interfaces. So, we can base WIS engineering on either the user world description, the systems prescription, the developers presentation specification, the developers systems specification.

We may extend the ladder by introduction layer, the deployment layer, and the maintenance layer. Since the last layers are often considered to be orthogonal to each other and we are mainly discussing WIS engineering the three layers are out of our scope.

5.3. Another Dichotomy for Web Information Systems Development

We thus develop another methodology for web information systems. WIS have two different faces: the systems perspective and the user perspective. These perspectives are tightly related to each other. We consider the presentation system as an integral part of WIS. It satisfies all user requirements. It is based on real life cases. The dichotomy is displayed in Figure 4 where the right side represents the system perspective and the left side of the ladder represents the user perspective.

Software engineering has divided properties into functional and non-functional properties, restrictions and pseudo-properties. This separation can be understood as a separation into essential properties and non-essential ones. If we consider the dichotomy of a WIS then this separation leads to a far more natural separation into information system requirements and presentation systems requirements. The system perspective considers properties such as performance, efficiency, maintainability, portability, and other classical functional requirements. Typical presentation system requirements are usability, reliability, and requirements oriented to high quality in use, e.g., effectiveness, productivity, safety, privacy, and satisfaction. Safety and security are also considered to be restrictions since they specify undesired behaviour of systems. Pseudo-properties are concerned with technological decisions such as language, middleware, operating system or are imposed by the user environment, the channel to be used, or the variety of client systems.

6. Extending the Triptych to the Software Modelling Quadruple

We are going to combine the results of the first three solutions into architecture development. One dimension of software engineering that has not been yet integrated well is the software architecture. Modelling has different targets and quality demands depending on the architecture. For instance, mainframe-oriented modelling concentrates on the development of a monolithic schema with a support by view schemata for different aspects of the application. Three-tier architectures separate the system schema into presentation schemata, business process schemata and supporting database schemata based on separation of concern and information hiding. Component architectures are based on
'meta-schemata’ that describe the intention of the component, the interfaces provided by the component, and the bindings among the interfaces. SOA architectures encapsulate functionality and structuring into services and use orchestration for realisation of business tasks through mediators. Therefore, application domain description is going to be extended by consideration of architectures and environments. Software architecture is often considered from the technical or structural point of view and shows the association of modules or packages of software. Beside this structural point of view we consider the application architecture that illustrates the structure of the software from the application domain perspective. Additionally we might include the perspective of the technical infrastructure, e.g. periphery of the system. These three viewpoints are one the most important viewpoints of the same architecture. We call an architecture documentation architecture blueprint.

Summarizing we find four interwoven parts of a software system documentation that we need to consider and that is depicted in Figure 5. The tasks and the objective of (conceptual) modelling changes depending on the architecture that has been chosen for the system.

6.1. The Prescription of Requirements

Architecture has an impact on development of early phases. We consider first requirements description.
Software engineering has divided properties into functional and non-functional properties, restrictions and pseudo-properties. This separation can be understood as a separation into essential properties and non-essential ones. If we consider the dichotomy of a WIS then this separation leads to a far more natural separation into information system requirements and presentation systems requirements. The system perspective considers properties such as performance, efficiency, maintainability, portability, and other classical functional requirements. Typical presentation system requirements are usability, reliability, and requirements oriented to high quality in use, e.g., effectiveness, productivity, safety, privacy, and satisfaction. Safety and security are also considered to be restrictions since they specify undesired behaviour of systems. Pseudo-properties are concerned with technological decisions such as language, middleware, operating system or are imposed by the user environment, the channel to be used, or the variety of client systems.

Properties are often difficult to specify and to check. We should concentrate on those and only those properties that can be shown to hold for the desired system. Since we are interested in proofing or checking the adherence of the system to the properties we need to define properties in such a way that tests or proofs can be formulated. They need to be adequate, i.e. cover what business users expect. At the same time, they need to be implementable. We also must be sure that they can be verified and validated.

6.2. Architecture-Driven System Development

WIS specification is often based on an incremental development of WIS components, their quality control and their immediate deployment when the component is approved. The development method is different from those we have used in the first layers. Application domain description aims in capturing the entire application based on exploration techniques. Requirements prescription is refining the application domain description. Specification is based on incremental development, verification, model checking, and testing. This incremental process leads to different versions of the WIS: demo WIS, skeleton WIS, prototype WIS, and finally approved WIS.

Software becomes surveyable, extensible and maintainable if a clear separation of concerns and application parts is applied. In this case, a skeleton of the application structure is developed. This skeleton separates parts or services. Parts are connected through interfaces. Based on this architecture blueprint, an application can be developed part by part.

We combine modularity, star structuring, co-design, and architecture development to a novel framework based on components. Such combination seems to be not feasible. We discover, however, that we may integrate all these approaches by using a component-
based approach [26,27]. This skeleton can be refined during evolution of the schema. Then, each component is developed step by step. Structuring in component-based co-design is based on two constructs:

**Components:** Components are the main building blocks. They are used for structuring of the main data. The association among components is based on ‘connector’ types (called hinge or bridge types) that enable in associating the components in a variable fashion.

**Skeleton-based construction:** Components are assembled together by application of connector types. These connector types are usually relationship types.

A typical engineering approach to development of large conceptual models is based on general solutions, on an architecture of the solution and on combination operations for parts of the solution. We may use a two-layer approach for this kind of modelling. First, generic solutions are developed. We call these solutions *conceptual schema pattern set*. The architecture provides a general *development contract* for subparts of a schema under development. The theory of conceptual modelling may also be used for a selection and development of an assembly of modelling *styles and perspectives*. Typical well-known styles [24] are inside-out refinement, top-down refinement, bottom-up refinement, modular refinement, and mixed skeleton-driven refinement. A typical perspective is the three-layer architecture that uses a conceptual model together with a number of external models and an implementation model. Another perspective might be the separation into an OTP-OLAP-DW system. The adaptation of a conceptual schema pattern set to development contracts and of styles and perspectives leads to a *conceptual schema grid*.

### 6.3. Architecture Blueprint

An architecture blueprint consists of models, documents, artifacts, deliverables etc. which are classified by the following states:

- **The architecture framework** consists of the information or data view, functional business or domain view, integration or data-flow view, deployment or technology view, and infrastructure or embedment view.

- **The WIS development architectures guide:** The current architecture is the set all solution architecture models that have been developed by the delivery projects to date. Ownership of the solution architecture models are transferred to the current Enterprise Architecture when the delivery project is closed. The development state architecture represents the total set of architecture models that are currently under development within the current development projects. The target vision state architecture provides a blueprint for the future state of the architecture needed in order to satisfy the application domain descriptions and target operating model.

### 7. Applying Architecture-Driven and Application-Domain-Ruled Modelling Methodologies

#### 7.1. The CottbusNet Design and Development Decisions

Let us consider the event calendar in an infotainment setting of a city information system. This calendar must provide a variety of very different information from various heterogeneous resources:
• Event-related information: Which event is performed by whom? Where from are the actors? How the event is going on?
• Location-based information: Which location can be reached by which traffic under which conditions with whose support?
• Audience information: Which audience is sought under which conditions, regulations and with which support?
• Marketing information: Which provider or supplier markets the event under which time restrictions with which business rules?
• Time-related information: Which specific time data should be provided together with events?
• Intention information: Are there intentions of the event that should be provided?

The event calendar is based on a different databases: event databases for big events, marketing events, sport events, cultural events, minor art events etc.; location databases for support of visitors of the event providing also traffic, parking etc. information; auxiliary databases for business rules, time, regulations, official restrictions, art or sport activists, reports on former events etc.

It is not surprising that this information is provided by heterogeneous databases, in a variety of formats, in a large bandwidth of data quality, in a variety of update policies. Additionally, it is required to deliver the data to the user in the right size and structuring, at the right moment and under consideration of the user’s information demand. Consider, for instance, minor art events such as a cabaret event held in a restaurant. The information on this event is typically incomplete, not very actual, partially inexact and partially authorised. The infotainment site policy requires however also to cope with such kinds of events.

We might consider now a number of architectures, e.g., the following one:
• **Server-servlet-applet-client layered systems** typically use a ground database system with the production data, a number of serving databases systems with the summarised and aggregated data based on media type technology [17], and playouting systems based on container technology [13] depending on adaption to the storyboard [18].

• **OLTP-OLAP-Warehouse systems** [11,12] use a ground database system for OLTP computing, a derived (summarised, aggregated) OLAP system for comprehensive data delivery to the user, and a number of data warehouses for data playout to the various kinds of users.

Depending on these architectures we must enhance and extend the conceptual schema for the different databases, the workflow schemata for data input, storage, and data playout to the user.

### 7.2. The Resulting Quality of Service and Tracking Back Problems to Decisions Made

Quality of WIS is characterized depending on the abstraction layers [8]:

**Quality parameters at business user layer** may include *ubiquity* (access unrestricted in time and space) and *security/privacy* (against failures, attacks, errors; trustworthy; privacy maintenance).

**Quality parameters at conceptual layer** subsume *interpretability* (formal framework for interpretation) and *consistency* (of data and functions).
Quality parameters at implementation layer include durability (access to the entire information unless it is explicitly overwritten), robustness (based on a failure model for resilience, conflicts, and persistency), performance (depending on the cost model, response time and throughput), and scalability (to changes in services, number of clients and servers).

We use a number of measures that define quality of service (QoS) for WIS:

- **Deadline Miss Ratio of User Transactions**: In a WIS QoS specification, a developer can specify the target deadline miss ratio that can be tolerated for a specific real-time application.
- **Data Freshness**: We categorize data freshness into database freshness and perceived freshness. Database freshness is the ratio of fresh data to the entire temporal data in a database. Perceived freshness is the ratio of fresh data accessed to the total data accessed by timely transactions - transactions which finish within their deadlines.
- **Overshoot** is the worst-case system performance in the transient system state. In this paper, it is considered the highest miss ratio over the miss ratio threshold in the transient state. In general, a high transient miss ratio may imply a loss of profit in e-commerce.
- **Settling time** is the time for the transient overshoot to decay and reach the steady state performance.
- **Freshness of Derived Data**: To maintain the freshness, a derived data object has to be recomputed as the related ground database changes. A recomputation of derived data can be relatively expensive compared to a base data update.
- **Differentiated Timeliness**: In WIS QoS requirements, relative response time between service classes can be specified. For example, relative response time can be specified as 1:2 between premium and basic classes.

We observe that these quality of services characteristics are difficult to specify in systems if architecture is not taken into consideration. Let us consider data freshness as an example for WIS. Data freshness results is related to information logistics that aims in providing the correct data at the best point of time, in the agreed format and quality for the right user with the at the right location and context. Methods for achieving the logistics goals are the analysis of the information demand, storyboarding of the WIS, an intelligent information system, the optimization of the flow of data and the technical and organizational flexibility. Therefore, data freshness can be considered to be a measure for appropriateness of the system. Depending on the requested data freshness we derive the right architecture of the system.

7.3. Resolution and Toleration of QoS Problems

Based on our co-design modelling approach and as a result separation of concern within the software engineering quadruple be can derive a number of techniques for architecture-driven and application-domain-rules modelling of high quality WIS:

- **Introduction of artificial bottlenecks**: Instead of replicating data at different sites or databases we may introduce a central data store that exhibits a number of versions to each of the clients that require different data.
Introduction of a tolerance model: We may introduce an explicit tolerance model that decreases the burden of data actuality to those web pages for which complete actuality is essential.

A cost-benefit model of updates: Updates may sometimes causes a large overhead of internal computing due to constraint maintenance and due to propagation of the update to all derived data. We thus may introduce delays of updates and specific update obligations for certain time points. Typical resulting techniques are dynamic adaptation of updates and the explicit treatment by an update policy.

Data replication in a distributed environment: Data access can be limited in networking environments. The architecture may however introduce explicit data replication and specific update models for websites.

This list of techniques is not complete but demonstrates the potential of architecture-driven WIS development.

8. Conclusion

This paper discusses the results of a project that was aiming in developing a methodological approach to web information systems development. Most approaches known so far did not take into consideration architectural issues. Typically, they are taken for granted or assumed on default. This paper shows that architectures have a deep impact on the development methodology. We took as an example web information systems development. These systems are typically based on the 2-tier architectures. The information system development part is very-well considered. The presentation system development is often mixed with the information system development. It cannot however be mixed. We separate these two systems from each other.

While separating we discover that in this case the application domain description fits very well with the support by the presentation system. This description is the source for requirements prescription. The later results in software specification and later development and coding of the system. The presentation system conceptualisation and coding can either be done before considering the information system or done afterwards.

Classical approaches consider the three facets of system development: application domain description, requirements prescription and software specification. We discover in this paper that there is a fourth facet that cannot be neglected: architecture of the system. Therefore, we extend the classical framework to the software modelling quadruple.

References

Acknowledgement

We would like to thank the Academy of Finland and the German Academic Exchange Service (DAAD) for the support of this research.