eGovWDF:Validation – A new approach to input validation in Web based eGovernment applications

WALTER KERN
University of Regensburg

In this paper we introduce the topic input validation, analyze its great importance to Web applications and suggest a new comprehensive approach to input validation. The approach has been developed as a result of an evaluation of current input validation approaches that showed that no sufficient solution to common input validation requirements is available at present. The paper describes important requirements for input validation frameworks, especially in the Web context, and introduces main concepts of this approach. The approach is based on the declarative, rule based definition of validation logic and the automatic translation of validation rules into server side and client side code. It supports conditional, composite and complex inter-field validation scenarios. It considers topics such as value normalization, inter-field dependencies and validation actions and integrates these aspects into a consistent validator based system. Our evaluation shows the benefits of this new approach and highlights its advantages compared to other popular and promising approaches such as PowerForms or Topes.

Keywords: Human-computer interaction, input validation, validators, composite validation, eGovWDF, data normalization, inter-field dependencies

1. INTRODUCTION

There has been a large growth of Web 2.0 [O'Reilly 2005] based Web sites and related technologies [Forrester Research Inc. 2008; Weinreich et al. 2008] due to the richness of graphical user interfaces, high interactivity and other potentials that Web 2.0 based applications offer [Paulsen 2005; Lawton 2008].

Because Web 2.0 applications usually provide more frontend elements for user supplied data than classic Web applications and because of the large influence and importance of invalid user provided data in respect of central Web application aspects, such as data integrity, security and user guidance (cf. section 2.2), input validation is an essential, intellectual and labor intensive task [Li et al. 2007].

There are many examples which illustrate the importance (cf. section 2.2) of input validation. Grete Mossbacks' case is probably the most popular one. Mrs. Fossback wanted to transfer $100.000 to her daughter, but transferred the money to an unknown person instead because of a simple keying error [Olsen 2008]. The recipient gambled away much of the sum before the policy was able to confiscate the remainder. Olsen [2008] conducted a survey with his students to analyze the common frequency of this kind of keying errors. His results showed that 7 percent of the account numbers specified by this test subjects were wrong. Furthermore in only 2.7 percent of the test cases the students recognized slightly manipulated transaction number errors on the confirmation screen, which undergirds the mandatory provision of input validation.

Although there are several approaches to a solution there is currently no comprehensive one which is able to satisfy all requirements (cf. section 3). As a consequence many applications do not provide validation at all, because programmers often drop input validation when inputs may appear in various formats or when validation criteria cannot be precisely specified [Scaffidi et al. 2008]. Moreover Hayes and Offutt [1999] explain that systems which depend on user provided data are mostly undocumented although they usually have to be maintained for several years.

This paper aims at introducing the main aspects of a new comprehensive approach to input validation which targets Web applications but is also usable in non-Web environments. The approach takes *Web accessibility* [W3C 1999; W3C 2008a] related requirements into account because the corresponding framework is part of *eGovWDF* (eGovernmental Web Development Framework), an experimental Web application framework featuring new approaches and scientific concepts at the intersection of Web 2.0 and Web accessibility. The research is conducted at the University of Regensburg and

implemented at the Landesamt für Finanzen, which is a regional branch of the financial authority of the Bavarian state. In the near future, our framework is intended to be used in a project called BeihilfeOnline which is a *Web 2.0* based application to acclaim refunds for expenses for medical treatments. This system will be implemented starting November 2010 and will finally be offered to all civil servants of Bavaria which includes about 480.000 users.

Because of the large extent of this new approach this paper acts as an introduction and focuses on the validation logic related aspects of eGovWDF. Further papers will deal with the validation visualization concepts, client side validation paradigms and the *eGovWDF Designer*, a tool which was designed to be used by the employer to define his (executable) validation requirements himself by a simple Drag & Drop supporting user interface.

2. DEFINITION AND RELEVANCE OF INPUT VALIDATION

The concept of *validation* is well known, for instance in the fields of simulation models [Pace 2003; Sargent 1987], user interfaces [Schlumberger 1989], resource structures [Balmin et al. 2004], complex system protocols [West 1989], distributed systems [Gao et al. 1995], software quality assurance [Nance and Arthur 1994] and input validation [Hayes and Offutt 1998].

According to Hayes and Offutt [1998] *input validation* refers to "functions in software that attempt to validate the syntax of user provided commands/information". Liu and Tan [2008] provide a slightly more generic definition by saying "Input validation is the enforcement of constraints that an input must satisfy before it is accepted in a program".

Summing up, *input validation describes the checking of user specified data with regard for the conformance to a formal specification*. Usually it is the aim of input validation to reject invalid user supplied data.

In the Web context, the definitions above have to be applied to Web applications. Therefore we define input validation as follows:

"Validation in the Web context means the automatic checking of user inputs specified in Web forms or as http(s) request parameters regarding their syntactic and semantic correctness, including the assistive visualization of validation errors of different severity to eliminate inconsistent and invalid inputs and to provide user guidance. "

Our definition does not only describe the process of checking user input for consistency and correctness but also includes the visualization of inconsistent and invalid data for the end user [Miller and Myers 2001]. Our definition is wider than others because we regard validation as comprehensive process that is based on human-computer interaction [Hewitt et al. 1992] and describes all aspects involved in the successful provision of user input for a computer application. Moreover, the definition includes user guidance aspects, because application workflows and navigation targets can be based on complex validation dependencies (cf. Section 4.4) and validation messages can serve as a kind of context sensitive help [Lee 1987].

Finally, Offutt et al. [2004] describe validation in the Web context by its syntactic and semantic specifics. *Syntactic correctness in HTML* can be expressed by built-in length restrictions (*maxlength* attribute of textboxes), input value restrictions (by varying the type of the html element, e.g. *checkboxes*), transfer mode (e.g. *GET* or *POST* requests), type of data access (e.g. hidden fields, cookies), input field selection and control flow restriction (action attribute of the html *FORM* tag). On the other hand, Offutt et al. [2004] identified data type conversion, data format validation and inter-value constraint validation as types of *semantic data input validation*.

2.1 Classification of input validation concepts

Because there is no commonly used vocabulary in the input validation domain, we will define some basic, general terms before we look into possible categorizations of input validation approaches.

Validation targets are the components whose values are to be validated, e.g. HTML input fields. The term *validation logic definition* describes the description of the characteristics of a specific type of input validation check, e.g. a *regular expression* [Ellul et al. 2005] based validation or *mandatory value* validation check. *Validation targets* and v*alidation logic definitions* are directly (e.g. by hardwired calls to validation methods with user input as parameters), or indirectly (e.g. declarative mapping of HTML input fields with validation logic definitions) associated. These associations are called v*alidation mapping definitions*. Input values can be of different relevance, e.g. a wrong password might terminate the current workflow (error) whereas an invalid email field may only result in a warning. This can be modeled by the concept of *validation severity (levels)*. *Validation triggering* means the enactment of validation of a single field or a group of input components. *Validation visualization* describes the indication of validation

related messages, e.g. info, warning and error messages. These messages aim at helping users in completing their data input tasks and are important elements of user guidance (the so-called *validation guidance*).

In view of the input validation classification, there are various possible categorizations based on different point of views or different central themes. According to our research [Kern 2008a] current input validation approaches are mostly differentiable by the following aspects:

- General aspects
  - Input validation can be conducted top-down (based on domain objects and object relational mapping – [Amber 1996]) or bottom-up (based on user interface input elements).
  - Validation can be implemented domain specific, e.g. for the Web context, or general to be usable for arbitrary platforms.
  - The Software architecture can be monolithic or modular. Validation logic definitions, validation mapping definitions, platform specific integration and validation error visualization can be implemented as separate layers or components [Nierstrasz et al. 1992], for instance.
- *Validation mapping* related aspects:
  - Validation mapping definitions can be formulated *programmatic* (e.g. classic if-statements), *component-based* (e.g. *ASP.NET Validators* – [Moore 2002]), *declarative and rule based* [Teraguchi et al. 2008], for instance.
  - Approaches may require the generation of *validation mapping definition specific code* or support a more *generic, run-time dynamic approach*.
- Validation logic related discriminators:
  - There are many ways of implementing validation logic, e.g. flow graphs [Christensen et al. 2003], parse trees [Buehrer et al. 2005], finite automatons [Wasserman and Su 2004], regular expressions and so on.
  - Validation solutions may provide a restricted, *sealed* set of validation logic or they may support the extension of existing validation logic definitions and the introduction of completely new ones.

- Validation visualization related discriminators:
    - Validation error visualization includes the design of validation error messages and behavioral characteristics for hiding and unhiding validation errors, e.g. showing validation errors on a separate area of the Web page or showing errors next to the validated component.
    - Web accessibility [W3C 2005; Kern 2008b] related aspects may be regarded or ignored.

In the Web context there are further aspects for differentiation. The most relevant ones are an immediate consequence of the request based client-server paradigm of Web applications [Fielding et al 1997]:

- Validation processing can be done either on the server side or on the client. Furthermore client side and server side validation can be combined. Additionally, AJAX technology [Garrett 2005; Zhang et al. 2007] can be used to trigger validation on the server side by client side code.
- Validation can be done immediate (e.g. on *keypress* or *onblur* event) or cumulative (e.g. on submit).
- Validation can be implemented *synchronous*, which means only on submit, or *asynchronous*, which means independent of submits and the regular request/response of the Web, e.g. on *onblur* events.

Besides these main discriminator aspects there are further aspects to categorize input validation approaches, e.g. *support of validation severity levels*, *differentiation between field-based and global validation*, tool support and *distributed versus local validation mapping definition* storage.

2.2 Relevance

The great relevance of error handling in software engineering is known for a long time and a lot of research has been dedicated to this topic [Shah et al. 2008] and customers are usually confronted with a lot of errors that can be expensive [Graunke et al. 2003].
Regarding the importance of input validation, our research [Kern 2008a] indicates there are three main reasons for the implementation of input validation:

- Data integrity
- Web application security

- User guidance

Data integrity [Sivathanu et al. 2005] means data is consistent, correct and complete. Generally, the validation of data constraints is one of the most important tasks of a system to ensure integrity [Laprie et al. 2004]. According to Liu and Tan [2006] input validation has always been essential for the control and accuracy of inputs to software systems with intensive user interaction. Aljawarneh et al. [2007] explain that there are three established techniques in common usage for ensuring the integrity of Web content: SSL, firewalls and *form-field validation* to protect against harmful data at the client- and server-side. In the Web context input validation is such important for data integrity because HTML forms in contrast to their designated but not widely supported successor XForms [Cardone et al. 2005] do not support to restrict textual user inputs to specific data types, formats or other constraints [Brabrand et al. 2000]. This means input validation is required because data integrity is not ensured by design. Furthermore even if forms based data constraints would be supported, input validation is still important, because manipulations on the client or corruptions of data transmissions are possible.

Web application security [Gollmann 2008] refers to the protection of Web applications and relating data. It deals with typical security threats such as cross-side scripting (XSS) [Jim et al. 2007] and privacy attacks [Jackson et al. 2006; Bortz and Boneh 2007]. Besides these scripting issues [Yue and Wang 2009], there are also main threads directly related to user inputs such as SQL injection attacks because most Web applications are database driven. According to Li et al. [2007] the invalidated input can be regarded as one of the most critical Web application security flaws. Microsoft Corporation [2003] approve the importance of input validation in view of Web application security by saying *"Proper input validation is an effective countermeasure that can help prevent XSS, SQL injection, buffer overflows, and other input attacks."*. Brinhosa et al. [2008] make this even clearer by stating that input validation attacks are becoming one of the most frequent attacks regarding Web Applications and Web Services. They also mention the ineffectiveness of traditional counter-measures such as firewalls against application level attacks because in contrast to application specific input validation these mostly external mechanisms lack of application specific information to know the meaning and legal values of the field and request parameters. A current study of the MITRE Corporation [2009] acknowledges the big importance of input validation for Web application security by saying that input validation is "the number one killer of

healthy software". This statement is detailed by significant values in the categories attack frequency, weakness prevalence and attacker awareness.

Regarding user guidance input validation can help users to improve their interaction with Web applications by showing them which information is requested and in which format. This can be considered as a kind of context sensitive help [Lee 1987; Nielsen 2005]. Furthermore, input validation - as kind of input assistance - is explicitly called for by *Guideline 3.3* of the *Web Content Accessibility Guidelines (WCAG) 2.0* [W3C 2008b]. Moreover advanced input validation systems may understand different representations of user input and may transform the user provided data to a format the Web application requires, e.g. consistent date format. This would allow the user to specify the data the way he knows it and minimize the need for corrections by the user. An intuitive and assistant validation mechanism can also help the user to fix type errors and to learn how to avoid the problems in the future. This is especially important if the user works with critical applications like online banking systems because type errors can lead to negative financial consequences. In summation, input validation can improve user satisfaction by preventing operating errors, teaching the most efficient and correct specification of data and by assisting users in the correction of input errors.

Additionally, there has been a large growth of Web 2.0 based Web sites and related technologies [Forrester Research Inc. 2008; Weinreich et al. 2008] due to the richness of graphical user interfaces, high interactivity and collective intelligence potentials that Web 2.0 based applications offer. Because Web 2.0 applications behave more and more like desktop applications they are usually more complex and as a consequence provide more frontend elements for user provided data which makes input validation even more important.

3. MARKET ANALYSIS

In this section we will focus on the requirements for input validation frameworks in the eGovernment context. After we show the sources of input validation requirements, we will detail the requirements with regards to validation logic and summarize the results of our market analysis [Kern 2008a].

3.1 Sources of input validation requirements

The requirements for an input validation framework in the eGovernment context that are stated in this paper are derived from German eGovernment related policies, laws and

recommendations and are based on concepts of modern software engineering and IT security. However, the orientation on German documents does not restrict the global universality of these requirements because most concrete documents in German eGovernment are derived from international ones. Especially in the field of Web accessibility the standards are almost equivalent, e.g. the requirements formulated in WCAG 1.0 [W3C 1999] and the concrete checklists of Germanys Web accessibility related law called “Barrierefreie Informationstechnik-Verordnung 1.0 (BITV)” (Decree on barrier free access to information technology) [Deutsches Bundesministerium des Innern 2002] are contentual identical [Kern 2008b]. Moreover, the major part of the requirements is more general and not specific to the eGovernment domain. In detail input validation requirements in the eGovernment context mostly arise from the following areas:

- eGovernment related laws, standards and recommendations
- International non-governmental recommendations such as WCAG
- Concepts of modern software engineering
- Web based systems
- IT Security

According to Kern [2008a] the requirements for a validation framework in the German eGovernment context can mostly be derived from the following documents:

The document “Standards und Architekturen für eGovernment-Anwendungen” (SAGA 4.0) [KBSt. 2008], which can be translated into “standards and architectures for eGovernment applications”, aims at providing a guideline and orientation aid for the forward-looking conception of technical architectures and IT applications. Primary objectives of SAGA are interoperability, reusability, openness, scalability and cost and risk reduction of eGovernment applications [KBSt. 2008].

V-Modell XT 1.21 [KBSt. 2006] is another document that describes a software development process and is provided by the KBSt. Additionally, V-Modell XT mentions several principles like reusability and component based architectures.

Besides, the German law for Web accessibility, BITV 1.0, defines a concrete checklist based on WCAG 1.0 [W3C 1999]. It applies to all federal agencies, but only to Web sites and Web applications and not to desktop applications. Because the successor of WCAG 1.0, WCAG 2.0 [W3C 2008a], was finished in 2008, this specification is also to be considered.

Moreover, IT Security related work and general concepts of modern software engineering such as Buschmann et al. [1996] and Pressman [1996] are relevant sources for validation requirements. In the Web domain validation topics such as support of client- and server side validation [Moore 2002] or context sensitive help [Lee 1987; Nielsen 2005] are most important aspects.

3.2 Focus on validation logic related requirements

As mentioned in the previous section, there are input validation requirements in different domains. Because of the scope of this paper we will detail the validation logic and software engineering related requirements below although our market analysis [Kern 2008a] is based on much wider set of requirements. These requirements which are not discussed here include validation visualization and platform integration aspects among other things [Kern 2008a]. In view of validation logic our analysis is based on the following requirements.

The separation of concerns and loose coupling are general goals [Fowler et al. 2002; Buschmann 1996]. However these paradigms are also advised by governmental documents such as SAGA 4.0 [KBSt. 2006]. SAGA 4.0 concretes these general suggestions by describing component oriented [Nierstrasz et al. 1992], service oriented (SOA) [Mahmood 2007] and multi tier architectures [Zhang et al. 2008] as recommended system architectures [KBSt. 2006]. Applied to the topic of validation frameworks, we demand the separation of validation logic formulation, validation integration and validation error visualization as a minimum to make the implementation of these different aspects interchangeable. The separation of validation logic formulation form validation integration also helps to define validation logic in a neutral representation and in consequence to use the uniform definitions on various platforms, e.g. in Web applications or desktop applications. This platform specific integration of validation logic is to be done by a validation integration layer or component. Because the average period of business logic usage is usually longer than the utilization of the user interface, the validation error visualization aspects are also to be implemented in an interchangeable way.

In view of validation logic notation, declarative and rule based formulation [Ligeza 2006] of validation logic are to be supported. One of many reasons behind this is that by using a declarative approach, validation logic can be defined at a higher abstraction layer instead of hardwiring validation checks in an imperative way. This also builds the

foundation for a framework whose validation rules can be flexibly combined and easily interchanged which also increases the SAGA 4.0 objective maintainability [KBSt. 2008; Broy et al. 2006]. It also simplifies the understanding of the validation logic and eases the implementation of tools to define rules in a more convenient way.

Moreover validation logic is to be reusable [Scaffidi et al. 2008]. In detail, we demand concepts such as inheritance [Biddle and Tempero 1996], combinability and scalability [Bondi 2000; D'Antonio et al. 2004] of validation logic as central requirements to achieve a certain minimum of reusability. Combinability is especially important because without combinability, it would be necessary to create a custom specific implementation for each specific input validation requirement which can lead to a combinatory explosion of implementations. For instance, if there are two distinct validation logic implementations for mandatory field and email validation and the combination is needed, it would be necessary to implement a new logic entity which includes both aspects instead of reusing the existing ones and combining them by declaration. Similarly, inheritance [Biddle and Tempero 1996] is useful to enhance reusability because this makes it possible to refine generic validation functionality and adapt it to the specific requirements, e.g. a regular expression logic rule can be used to create an email address format validation rule. In addition, data value and form control dependencies are to be supported [W3C 2001], because it is a common requirement to have components whose validity depends on the state or validity of other components.

Although scalability [Cheng 1994] is important for reusability, it is also vital for the practicalness of the corresponding validation framework because for some scenarios it may be sufficient to save validation logic in local file system and for others databases or distributed data stores such as Clouds [Sedayao 2008] may be required. As a consequence we demand a validation framework that allows saving the validation logic definitions in an arbitrary number of arbitrary distributed data storages which can be achieved by using plugin [Fowler 2002] concepts. Scalability is also important for increasing reliability and in combination with inheritance to make it possible to build a hierarchical system of validation policies such as companywide definitions, application specific peculiarities or something in between.

Furthermore validation logic should be extensible because it is not possible to foresee future requirements or deliver a solution that satisfies all – even domain specific – needs. In this connection plugin concepts, factory pattern approaches or similar mechanisms [Gamma et al. 1994] are of importance.

Because of economical restrictions, legacy applications are often enhanced rather than newly developed. This trend is even stronger in the eGovernment sector because of a possibly more conservative thinking and demographic influences [Jauvin 2007]. As a consequence, a new approach should be downward compatible and support an unobtrusive as possible extension of legacy applications to find acceptance. We advocate a bottom up approach (cf. section 2.1) that applies application-external validation definitions to user interface input elements instead of requiring a specific persistence framework. This allows keeping legacy applications that are developed with a specific persistence framework or without any persistence framework at all and leveraging rule based validation at the same time.

In addition, we demand a kind of plugin concept for the description format of validation logic because formats evolve or are replaced over time, but central business or validation logic often prevails for a long time. Thus, this requirement ensures long-lasting usability of validation logic definitions because the validation logic description format can be adapted to future needs without having to reimplement the logic itself.

Because validation errors can be of different levels of seriousness [Williams 2004], an input validation framework should also support a way to specify the severity of validation rules. Different severity levels are important because depending on the specific business process some errors may be critical and process-terminating and some others may be only informative or warning. The severity setting should be applicable to the mapping between the validation targets and the validation logic rules because otherwise it would be necessary to create duplicates of validation logic rules for different severity requirements in different applications.

In the Web context, duality of validation on both the client and on the server is postulated [Moore 2002]. More precisely, a validation framework shall support a single declarative definition of validation logic and validation mappings definitions and a transparent validation on client and server side based on these definitions without requiring any manual translation work by a human. The reason for the support of client side validation is its support for incremental validation and its advantage of avoiding server side round trips because no submit is required to do server side validation [Brabrand et al. 2000]. This also improves responsibility and leads to a decrease in network traffic.

3.3 Methodology and Evaluation

The evaluation was carried out in terms of an expert review. We decided to introduce three levels of conformance to the requirements in order to make the results comparable:

- 0.0: No support of a requirement.
- 0.5: Partial support of a requirement.
- 1.0: Complete support of a requirement

This conformance level is determined for every single (weighted) validation requirement. As a result, we get detailed data about the fulfillment of every single validation requirement which forms the basis for further evaluation [Kern 2009]. We calculate the absolute overall value and the relative degree of fulfillment to make comparisons based on single, scalar values possible. In this section we focus on showing the basic results of our evaluation [Kern 2008a] to explain the need for a new approach to input validation. A more detailed analysis that compares our approach (cf. section 4) to the approaches tested in [Kern 2008a] is provided in [Kern 2009] and discussed in section 5.

In our evaluation [Kern 2008a] we tested the following frameworks/concepts:

- Novel approaches
  - XForms validation
- Imperative approaches
  - Adobe Flex validation
  - WebDynpro validation
- Component based approaches
  - ASP.NET validation
  - JSF RI validation
  - Apache MyFaces Trinidad Converters and Validators
- Declarative approaches
  - Spring.NET validation
  - .net Validation Framework
  - Apache Commons Validator Framework
  - Hibernate Validator with Seam integration
  - XWork Validation Framework
  - LINQ with ASP.NET validation

The framework selection is based on preliminary research regarding the most important and promising validation frameworks in the Web field. Importance and

potential are derived from market power of the producer and market penetration of the framework. In the following we illustrate the rationale for our selection based on short framework overviews with focus on validation.

XForms [Cardone et al. 2005] is developed in tandem with XHTML and was chosen because at the time of evaluation it was regarded as novel approach that could replace HTML forms in the mid-term or long-term. XForms is based on the MVC architecture and supports data type specific user interface widgets which improves quality of user provided data and reduces the need for validation. Additionally, validation can be declaratively implemented on the client side without the need for JavaScript.

Adobe Flex [Wang 2009] provides technology and tools to create rich internet applications. Because of its novelty and its focus on rich internet application development rich validation was assumed. Furthermore this framework has great potential to flourish because it is based on flash technology which is available on 98% of all client systems.

WebDynpro [Cristea and Prostean, 2009] is a web application framework of SAP. It is based on a MDA approach and is part of the evaluation because of the market power of its producer and the support of domain object focussed data type constraints which allow validity constraints on data model level.

Active Server Pages .NET (ASP.NET) [Moore, 2002] and Java Server Faces Reference Implementation (JSF RI) [Burns and Kitain 2006] are both component based Web development frameworks. Both support so-called validation controls, which validate the values of assigned input controls. ASP.NET is provided by Microsoft which means strong market power. JSF RI is the reference implementation of the JSF specification provided by Sun. Because of the great influence of Sun and the wide implementation of Java there is great potential for JSF to expand. Apache MyFaces Trinidad Converters and Validators is based on JSF RI and extends its server side validation capabilities with client side validation functionality.

The remaining frameworks are further well-known solutions that support user input validation [Kern 2008a]. More details concerning framework selection and evaluation execution are mentioned in [Kern 2008a].

3.4 Discussion

Figure 1 shows the degrees of requirement fulfillment of all tested frameworks regarding validation logic, validation integration, validation visualization and further aspects. Regarding validation logic related requirements, the blue bars are to be considered. The

blue bar representing the XWork validation framework shows that the XWork validation approach achieves the highest degree of requirement fulfillment at about 50%. On average, a validation logic related degree of requirement fulfillment of 37% is achieved.

Considering all, including non validation logic related requirements, the best framework achieves at about 59% of requirement fulfillment whereas on average 51% of requirement fulfillment is achieved (figure 2).

This shows that currently there is no solution that is able to appropriately satisfy common requirements to input validation in the eGovernment domain. Moreover, the symbiotic combination of validation frameworks is not always possible because of different and incompatible concepts, platforms and APIs. Furthermore current scientific approaches (cf. section 6) also have several shortcomings that make them incapacitate to satisfy the requirements for input validation frameworks in the eGovernment domain. All this undergirds the necessity of more research and a new approach to input validation.

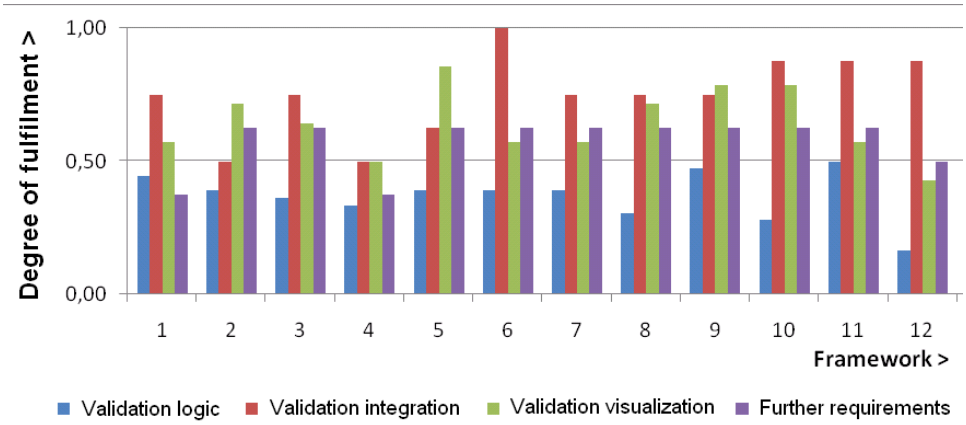

Fig. 1. Category specific fulfillment of validation requirements (%).

Legend:

- Framework 1: XForms validation
- Framework 2: ASP.NET validation
- Framework 3: JSF validation
- Framework 4: Adobe Flex validation
- Framework 5: WebDynpro validation
- Framework 6: Spring.NET validation
- Framework 7: .net Validation Framework
- Framework 8: Apache MyFaces Trinidad Converters + Validators

- Framework 9: Apache Commons Validator Framework
- Framework 10: Hibernate Validator + Seam-Integration
- Framework 11: XWork validation framework
- Framework 12: LINQ with ASP.NET

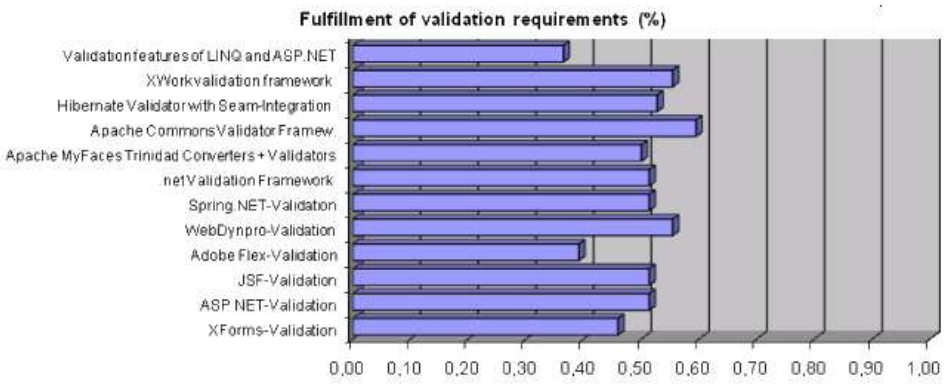

Fig. 2. Overall fulfillment of validation requirements (%).

4. THE EGOVWDF VALIDATION APPROACH

The eGovWDF validation framework was designed based on the requirements stated in Kern [2008a] and in section 3.2.

eGovWDF features an extensible, rule based, declarative, bottom up approach (cf. section 3.2) to input validation. *Basic validation logic definitions* (e.g. regular expression validation) are implemented in code and based on a common *IValidator* interface (cf. section 4.1). Concrete, *task-specific validation logic definitions* are *rule based* and comprised of references to basic validation logic definitions or other declarative definitions. These rule based, application independent definitions are stored in *policy definition resources*, which are referenced by *application definition resources or other policy definition resources*. Application definition resources contain application specific policy definitions and validation mapping definitions. Validation logic definitions and validation mapping definitions can be distributed across an arbitrary number of resources which improves scalability (cf. section 4.7). Moreover, both of them are compositional and support the specification of predefined and custom severity levels (cf. section 3.2). Applications integrate validation logic by a local configuration file which contains references to an arbitrary number of validation definition resources. Figure 3 depicts the different resource types and shows their connection.

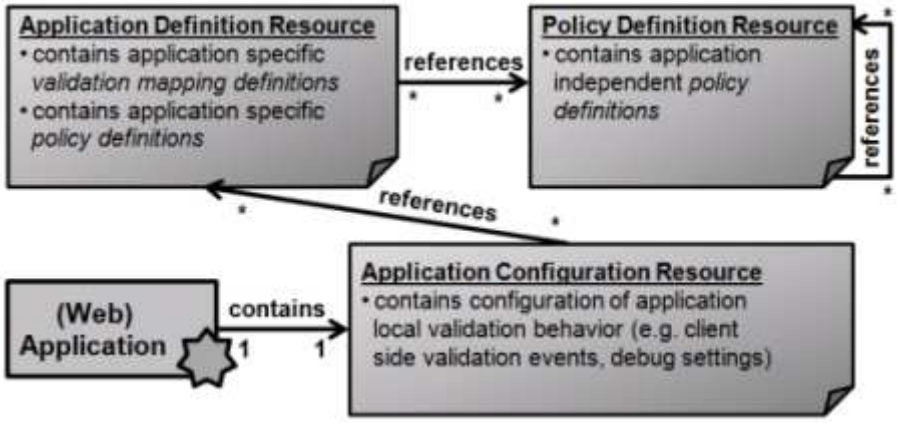

Fig. 3. Simplified view of the validation resource types.

The framework also supports an extensive inheritance model (cf. section 4.7) for validation logic and validation mappings. This can be used to define an arbitrary deep hierarchy of validation policies. For instance, there could be corporate policies and application specific ones that override the corporate ones depending on the chosen inheritance directives.

The format of validation policy definitions, validation mapping definitions and the data storage access are implemented by a plugin concept, which makes it possible to basically save validation logic and mappings of any format, e.g. XML, in any resource such as databases or the file system. This flexible resource access was implemented by a novel, generic approach to resource acquisition and processing called *Dimension Architecture* [Kern et al. 2010].

Finally, this approach supports the rule based uniform specification of validation logic and the transparent in-memory translation into validation checks on the server side and client side (cf. section 4.8).

In the following sections we will focus on the main concepts behind eGovWDF validation logic.

4.1 Validators

Validation logic definitions (cf. section 2.1) are implemented by *validators*. A validator encapsulates task type specific and usually interdisciplinary validation logic, e.g. validation of regular expressions. Validators are configured by an associative array of *validation parameters*, a *validation message* to display if validation fails and a *severity setting* (e.g. error, warning). A validation parameter can contain a simple configuration

value, e.g. the pattern for a *regular expression validator*, or a list of child validators which allows hierarchical nesting. Figure 4 shows the corresponding interfaces for validators and parameters.

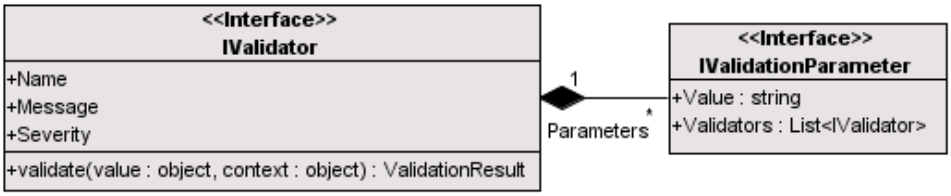

Fig. 4. Simplified view of the Validator and and IValidationParameter interfaces.

Validation logic is formulated by declarative validation rules based on references to validators (cf. section 4.2). All *validator related interfaces and concrete validators are implemented in server side and client side (JavaScript) code.* The complete validation API on client side and on server side is identical which enables a rapid learning curve for framework developers. This combination of client side and server side validation and central declarative validation definition has several reasons:

- Declarative non-imperative validation logic can automatically be transformed into client side *and* server side code without requiring the developer to understand any server side (e.g. Java) or client side (e.g. JavaScript) programming languages. It is our intention to allow the non-IT people that provide the functional validation requirements to implement their validation logic by themselves to minimize knowledge transfer issues between validation logic specification and implementation. This is similar to the support of end user programmers [Montgomery and Daniel 2009] by the validation approach of Scaffidi et al. [2008].
- Validation on the client side enables field-based, incremental validation and helps in avoiding roundtrips to the server side because submits can be avoided if client side validation fails [Brabrand et al. 2000].
- Because JavaScript may be disabled, manipulated or not supported by the client's browser, pure client side validation as depicted in Brabrand et al. [2000] is not sufficient. As a result this approach enforces server side validation, independently of validation on the client. This mandatory support of server side logic is also necessary for Web accessibility, because accessible Web sites have

to work independently of the scripting capabilities of the client browser [W3C 1999; Kern 2008b].

As depicted in Figure 5, there are four fundamental types of Validators:

- *Verification validators* check values or states of components, e.g. regular expression validators. Depending on the validation result and the validation severity the running process can be aborted.
- *Correction validators* normalize values (cf. section 4.6). Their application does not result in validation errors or warnings in contrast to *verification validators*. The intention of correction validators is to get a consistent representation of data for further processing and validation which is relevant regarding data integrity. Correction validators also aim at helping users in handling the application.
- *Composite validators* enable the composition of multiple validators, e.g. a logical *And validator* would have a validation result of true if the validation results of its nested *regular expression validator* and *value length validator* are true. *Composite validators* make it possible to build a tree of validators and to satisfy complex validation requirements like conditional validation and inter-component dependencies (cf. section 4.4). This is essential to achieve real flexibility, extensibility and reusability.
- *Action validators* encapsulate common actions that are to be executed depending on the validation result of an assigned verification validator. For instance, an action validator could show or hide a configurable component on the current Web page if the validation result of a *mandatory field validator*, which is assigned to a specific input field, is valid.

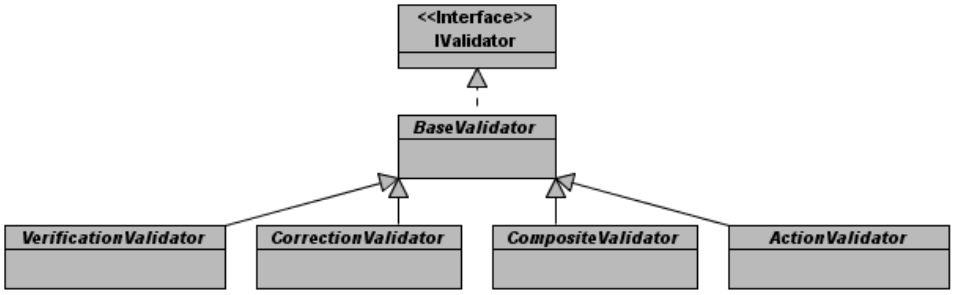

Fig. 5. Fundamental Validator types.

This approach supports four fundamental types of validators because error detection is only one part of a comprehensive validation process that is to be based on human-computer interaction [Hewitt et al. 1992]. According to De Paula et al. [2005], the correction of statements and the prevention of problems are an inherent part of human conversation which we generalize regarding user actions. Whilst Verification validators show errors to enable their correction by the user, Correction validators help to avoid the need for users to manually correct many errors. This diversification of Validator types combined with conventional input field descriptions and restrictions also correlates with the error handling categories of the Modeling Language for Interaction as Conversion (MoLIC) [De Paula et al., 2005]:

- Passive Prevention (PP): documentation or online instructions designed to prevent errors (e.g. the data format of a field) -> conventional input field descriptions, e.g. tooltips
- Active Prevention (AP): active mechanisms that prevent errors (e.g. input fields which restricted choice) -> conventional input field restrictions, e.g. drop down list with prespecified values to select from
- Supported Prevention (SP): asking the user to decide the reaction to an error condition (e.g. showing a warning which the user can ignore or take into account by changing the input data) -> Verification validator with non-terminating severity level
- Error Capture (EC): errors which are detected by the system and must be notified to users, but for which there is no undo option (e.g. file corruption) -> Verification validator with process-terminating severity
- Supported Repair (SR): informing the user about an error and allowing him to correct it (e.g., presenting an error message and the input the user has specified previously) -> Verification validator

As mentioned above, in addition to these error handling options our approach supports not only the manual correction of input errors by the users, but also their automatic correction.

In our approach, the validate function of a Validator executes the validation of the specified value and returns a result of type *ValidationResult*. A *ValidationResult* contains the error message and severity of the first matching validator that returned a negative validation result. It also includes the new value in case the value was altered by a

correction validator. It also contains a value which indicates whether the validation result should have influence on running processes because its default behavior is to stop the running process on validation errors of error severity.

It is possible to implement new validators but usually this is not required because validation logic can be expressed by configuring the provided general purpose validators, e.g. a *regular expression validator* can be configured to validate email addresses based on a parameter for a regular expression pattern. Even very complex validation scenarios can be implemented without the need for programming by using compositional and conditional validators (cf. section 4.3) in order to combine several validators and to even implement validation dependencies (cf. section 4.4).

4.2 Policies and mappings

As mentioned above, basic validation logic definitions are implemented by validators. However validators cannot be directly assigned to user interface components. Instead, user interface components are mapped with *policies* which are declarative configurations of validators. This decision was made to improve reusability and to ease the use of validation logic definitions by providing a more functional representation than the underlying technical general purpose validators. This decision also supports our goal of support for the delegation of the task of validation logic definition and mapping to the (non-IT) people that define the functional (validation) requirements. As mentioned avoe, this can prevent knowledge transfer issues between validation logic specification and implementation. Listing 1 shows a policy named *EmailPolicy* which customizes a *RegexValidator* by specifying its *Pattern* parameter to implement validation of email addresses.

```
<Policy name="EmailPolicy">
  <Validatortype="eGovWDF.Validation.Core.Validators.Common.RegexValidator"
    message="Invalid email address">
    <Param name="Pattern" value="([a-zA-Z0-9_\-\.]+)@((\[[0-9]{1,3}\.[0-
      9]{1,3}\.[0-9]{1,3}\.)|(([a-zA-Z0-9\-]+\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\]?)"/>
  </Validator>
</Policy>
```

Listing 1. *EmailPolicy* definition in *PolicyDefinitions XML format (ValPDXF).*

As mentioned above, *validation mapping definitions* associate the components to validate (*validation targets*) with validation policies. At runtime a component of the Web page called *ValidationManager* (cf. section 4.8) extracts the values of the validation targets and passes the values to the validators of the policies that are specified in the corresponding mappings. Listing 2 shows an example of some mappings which can either be *global* or *local*.

```
<Mappings>
 <Global>
  <ValidationMappings>
   <Component id="txtId" validationPolicy="AccountIdPolicy" />
  </ValidationMappings>
 </Global>
 <Local>
  <Context id="default2.aspx">
   <ValidationMappings>
    <Component id="txtUrl" validationPolicy="UrlPolicy" />
    <Component id="txtMail" validationPolicy="MailPolicy" />
   </ValidationMappings>
  </Context>
 </Local>
</Mappings>
```

Listing 2. Basic Validation Mappings sample *in ApplicationDefinitions XML format (ValADXF).*

Global mappings of an application definition resource apply to the whole application that references the resource. In the Web context this means all components on all Web pages with the specified ID are checked against the specified policy, e.g. all components with id *txtID* are checked against the policy *AccountIdPolicy*, for instance. *Local mappings* are context specific which means the current values of the corresponding components are only validated if the associated Web page equals the current Web page. For example, in Listing 2 the component *txtUrl* is only validated if the current Web page is *default2.aspx*. The interpretation of the platform independent definition format for mappings depends on the chosen *ValidationManager* (cf. section 4.8) component which

makes it possible to use the identical validation mapping definitions for different application user interfaces on different platforms.

4.3 Composite and conditional validation

In contrast to regular component based validators such as ASP.NET validators [Moore 2002] or JSF validators [Burns and Kitain 2006], the declarative approach of eGovWDF supports the combination of arbitrary validators by using *composite validators*, e.g. *AndValidator*, *OrValidator* and *NotValidator*. These basic validators are sufficient to implement every possible Boolean expression because it is possible to nest composite validators indefinitely. Figure 6 and Listing 3 show an example which is based on a composite validator. The policy ensures that a value tested against the policy is only valid if the value is specified (*RequiredValidator*) and numeric (*RegexValidator*) at the same time.

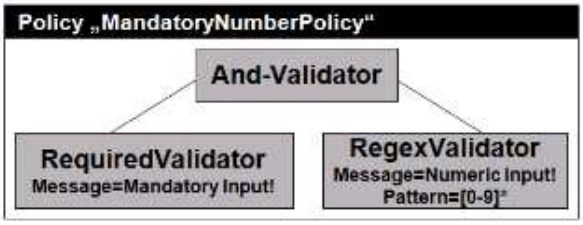

Fig. 6. Composition of validators.

```
<Policy name="MandatoryNumberPolicy">
 <Validator type="eGovWDF[…]OrValidator">
  <Param name="Operands">
   <Validator type="eGovWDF[…]RequiredValidator" message="Mandatory input!"/>
   <Validator type="eGovWDF[…]Common.RegexValidator"
     message="Numeric input!">
    <Param name="Pattern" value="[0-9]*"/>
   </Validator>
  </Param>
 </Validator>
</Policy>
```

Listing. 3. Declarative Validator Composition.

This idea gets even more flexible by using a validator of the type *PolicyReferenceValidator* which allows referencing other policies from within a policy.

For instance, instead of using a *RegexValidator* in the sample presented in Figure 6 the reusability could be enhanced by outsourcing the *RegexValidator* definition into a new Policy called *NumberPolicy*. This new Policy can be referenced from within a *MandatoryNumberPolicy* if a *PolicyReferenceValidator* is used instead of the *RegexValidator*. This type of reference is called *inter-policy reference* and illustrated in Figure 7 which shows the revised *MandatoryNumberPolicy* definition.

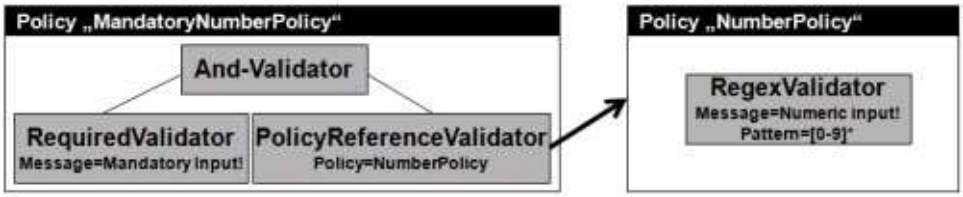

Fig. 7. Referencing policies within policies.

Furthermore *conditional validation* is supported by the *IfValidator* composition validator. The *condition parameter* as well as the *then parameter* supports a validator as argument which enables the execution of one validator in dependence of another validator and avoids the need for code to express conditions as most other solutions require [Kern 2008a]. It can also be configured whether the validation result of the condition validator is to be factored in the calculation of the validation result. In the example depicted in Figure 8, the value of the component A is only valid if it is

- numeric and a valid currency
- or non-numeric and equal to the hyphen character.

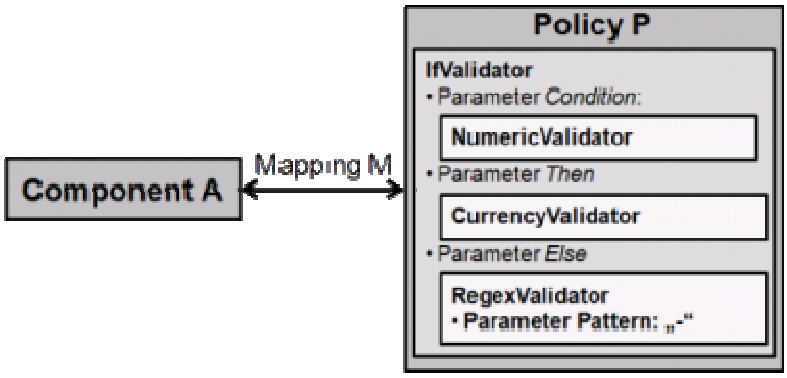

Fig. 8. Implementing basic conditions with the *IfValidator*.

More complex conditional expressions can be formulated by *nesting IfValidator* validators.

4.4 Component dependencies

Section 4.3 showed eGovWDF's support for composite and conditional validation and introduced basic concepts of inter-policy references which deal with the referencing of reusable policies from within other policies.

A more complex validation topic is *inter-component dependencies* which means the validation of a component can depend on other ones. This can be implemented by a combination of the *IfValidator* and the *ComponentReferenceValidator* which supports the specification of the id of another component. All nested validators of the *ComponentReferenceValidator* automatically get and validate the value of the component with the specified id instead of the component that is mapped in the corresponding policy. For example, the *RequiredValidator* in the *then branch* of the policy depicted in Figure 9 is only processed if the *ComponentReferenceValidator* in the condition branch evaluates to *true*. It evaluates to *true* if its nested validators, in this case another *RequiredValidator*, return *valid* as result of the validation of the referenced component A. This means component *A is valid*, if component *A and B are non-empty* or if component *B is empty*.

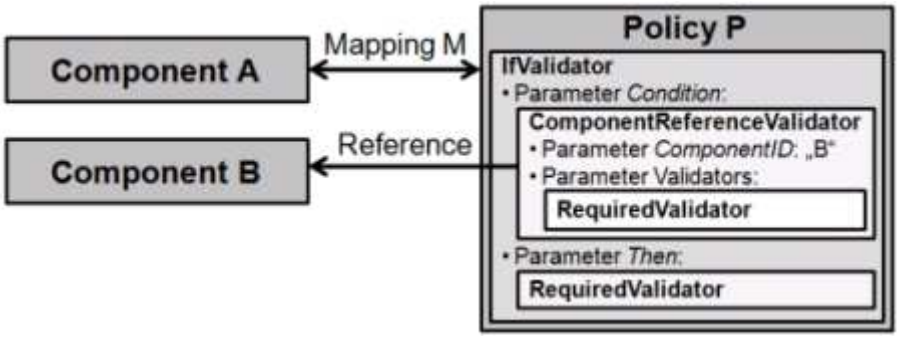

Fig. 9. Implementing inter-component references with the *ComponentReferenceValidator*.

By additionally using a *PolicyReferenceValidator* it is possible to not only reference the value of another component, but also reference a specific or all policies of another component which is called *inter-policy reference*. This can be useful in case the validity of one component is not only dependent of the value, but from the overall validity of the other component. However, policies that do not reference a policy but a concrete component are only allowed in application definition resources and not in policy

definition resources because references to other components in policy definition resources would weaken the reusability of application independent policy resources.

Because *dependency validators support inter-value and inter-policy references* this approach is very flexible. The values of referenced components can be validated against the policies of the referenced component or against any other policy or validation logic.

However the flexibility of referential Validators also brings some possible drawbacks such as *circular references* and *infinite loops* because a policy can reference itself or other policies which reference the policy. The definition of such validator cycles can be *prevented at policy design time and at runtime*. In science there is many research conducted in the field of cycle detection algorithms, e.g. [Nivasch 2004], [Boukerche and Tropper 1998] and [Schall 1990]. Because in most cases there will only be simple 1-dimensional requirements, e.g. email address or URL format, a simple adaption of the depth first search algorithm in regards to saving the visiting state by different colors based on Kamil [2003] is used. The cycle detection check based on this algorithm is conducted during *validation logic setup* at runtime and at design-time by our *graphical validation configuration tool* to prevent invalid definitions in advance.

4.5 Validation actions

Section 4.3 and 4.4 described the usage of validation dependencies to execute the validation of a component based on the validation result of other components. Though, there may be situations when it is required to execute actions depending on the validation result of one or more components. This is implemented by *action validators* which execute task type specific actions when executed. eGovWDF provides a basic set of general purpose action validators, e.g. a *VisibilityActionValidator* that changes the visibility of the validation target. By using a *ComponentReferenceValidator* (cf. section 4.4) it is possible to *change the visibility of another component* than the validation target. In the example depicted in Figure 10, the visibility of component B is set to true if validation of Component A is successful. The basic set of action validators also includes a *ValidationActionValidator* whose execution triggers the validation of a component.

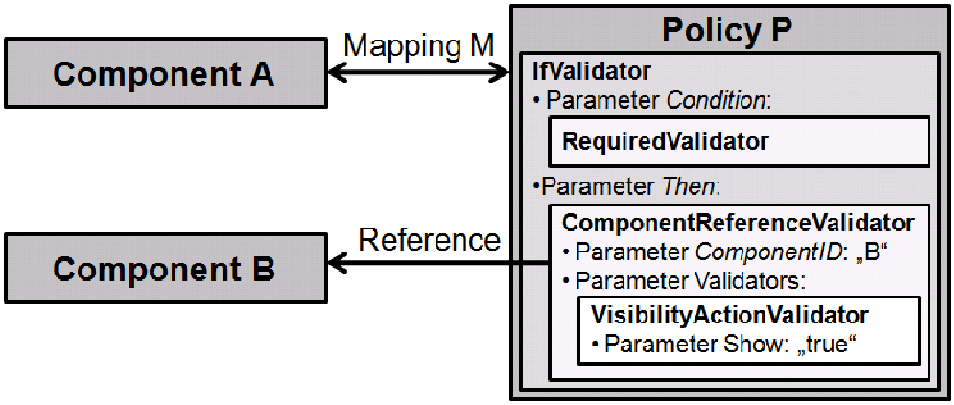

Fig. 10. Usage of *action validators*.

4.6 Value normalization

As stated in section 2.2, validation is primarily important for data integrity, Web application security and user guidance. However often there are multiple valid representations of data, e.g. country specific date formats. Since different information representations make it more complex and error-prone to process data further, this concept supports a specific type of validators which normalize information representation. These correction validators also make it possible for users to enter data the way they prefer and support the indication of the correction to help users in understanding the required format by doing. For example, a *DateCorrectionValidator* could be used to convert user inputs with different date representations, e.g. dd.MM.yyyy to UTC format (Figure 11). According to Scaffidi et al. [2008] normalization also helps in improving validation accuracy and reusability. By using composite validators, correction validators can be combined with and executed before verification validators to provide a consistent view of data before the actual verification takes place.

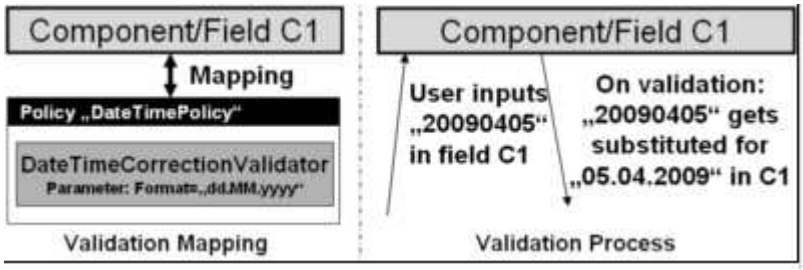

Fig. 11. Correction validator mapping and validation processing.

In contrast to several other approaches such as Scaffidi et al. [2008], the correction validator based approach does not concentrate on a single mechanism to decide which representations are equal. Instead, the mechanism for the transformation depends on the chosen correction validator, which allows a more problem specific and potentially a more efficient transformation, because the optimal correction validator can be chosen depending on the data field. For example it can be considered to be more transparent and problem specific to use a simple *DateCorrectionValidator* that transforms all recognized date formats to a specific and configurable date format than a more general text based *RegexCorrectionValidator* which allows capture group based substitution and aims at more complex transformations. In summation, the correction validator concept allows the use of the transformation logic that fits the data best.

4.7 Inheritance mechanisms

One major advantage of this approach is its wide support for *inheritance* [Taivalsaari 1996]. *Vertical distribution* (Figure 12) supports inheritance of default definitions and allows overriding or extending inherited definitions. The non existing limit of validation resources referencing other validation resources maximizes reusability and separation of concerns, e.g. companywide validation rules to application local validation rules. The horizontal distribution (Figure 13) on the other side optimizes scalability because validation related definitions can be simply separated into an arbitrary number of resources on the same hierarchy level.

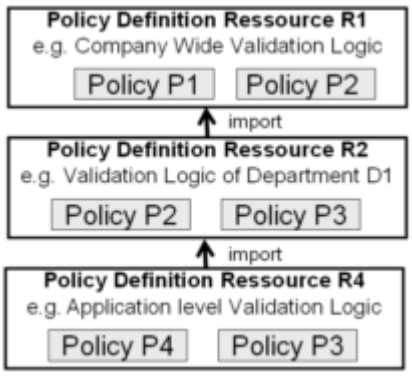

Fig. 12. Vertical distribution (classic inheritance; for increasing reusability).

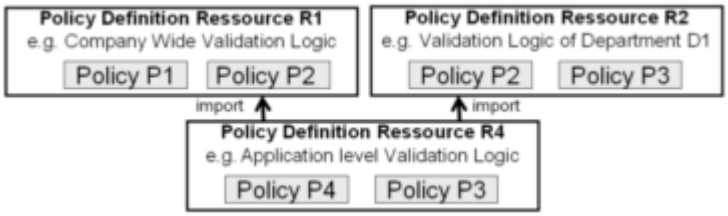

Fig. 13. Horizontal distribution (for maximizing scalability)

As mentioned above, policies can be referenced from other policies. This allows extending existing policies with custom sub organizational or even application specific logic by adding additional Validators or existing Policies.

If *horizontal distribution* is used, *multiple inheritance* can be achieved which could result in name collisions [Singh et al., 1995] if duplicate policies are to be imported. In this case it is not automatically decisive which policy has precedence if multiple policies with the same id are available. In contrast to multiple inheritance in object oriented programming languages, we decided to support the flexibility of specifying the application behavior if duplicates of policies and as a consequence name collisions are existing. This means, a name collision does not necessarily mean an application failure; instead the policy in the importing resource decides which one of the imported policies is to be used. .

eGovWDF supports the following main inheritance options to handle duplicates:

- Keep
- Override
- Join
- MergeException
- Custom

Keep is the default setting and makes the importing policy override the policy duplicates from the imported policies. This means the importing policy is kept which is the default behavior one is used to from inheritance in object oriented programming.

Override makes the last imported policy override all previous imported ones. This means if policy P is imported from policy resources R1 and R2 into policy resource R3, the last imported policy from R2 overrides the namely identical policies from R1 and R3, because according to figure 14 R2 is imported after R1. If R1 is to override R2 and the identical Policy from R3, the import order can be switched.

Join in the most *intelligent* option because it *merges* the importing policy with the policies with the same name from the imported policy resources. This means, if there is a resource R3 that imports policies from R1 and R2, a new policy is created which contains the combination of the policies with the same name from R1, R2 and R3. For example, if a *PasswordPolicy* at global level says a password has to be at least of length 6 and an application specific policy with the same name says a password has to contain at least 3 special characters, the new policy resulting from a *Join* will comprise of a logical *AndValidator* that contains the two policies. Figure 14 illustrates the three main inheritance options mentioned above.

MergeException is a very conservative option that throws an exception if two policies with the same name are found. This is useful in scenarios where you know that there are no duplicates allowed.

However there can be situations which require a more explicit and direct way to select the dominant policy. This is supported by the option *Custom* which allows to directly specify the resource whose policy is to be chosen (Format *Resource:Policy*). If this flexibility is not enough an existing policy can be referenced from within another policy and combined with any other validator by using composite validators.

All of these inheritance options also apply to validation mapping definitions at mapping element level and to any validation entity in general. Validation contexts (in the Web domain Web pages) support further inheritance settings. However, validation mapping definitions are additionally assigned to a context, e.g. Web page. At context element level several additional container related inheritance options are supported.

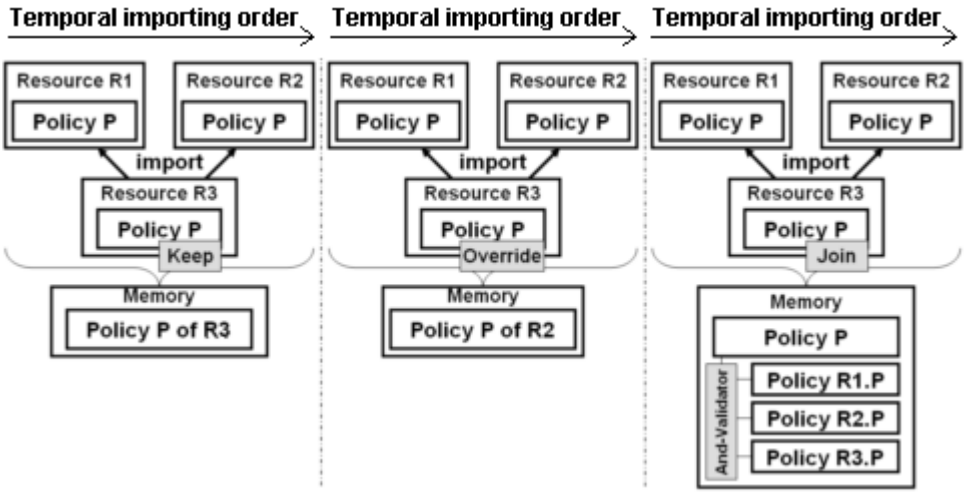

Figure 14. The inheritance options Keep, Override and Join.

4.8 Validation processing and platform integration

ValidationManager components are platform and framework specific and integrate validation into platforms and frameworks, e.g. the *AspNetWebValidationManager control* integrates validation processing into ASP .NET based Web applications. It's the job of the *Plugin Pattern* [Fowler et al. 2002] based *ValidationManager* components to setup client side validation handlers, to extract the values to validate from the current context (Web page) and to enact validation of all or specific values of the context by checking the values against the associated policies. Furthermore, *ValidationManager* components can cancel operations on validation errors with specific severities. In the default configuration, the *AspNetWebValidationManager* aborts submits on the client side if validation errors are present, for instance. On the server side it cancels the execution of event handlers if validation errors are present.

As mentioned in section 4.1, because validation policy definitions and validation mapping definitions are declarative and the client and server side validation interface is identical, validation *logic has to be only defined one time in a uniform format*. The central validation definitions can automatically and transparently be applied at all tiers (client and server side). According to Yang et al. architectures like this bring several key advantages such as decreased development time, increased simplicity [Yang et al. 2007] because of no need for logic partitioning between the tiers and better maintainability because of eliminating re-partitioning issues.

Our validation presentation framework uses this dual behavior to provide client side, field based live validation when the user enters data. For instance, if the user enters data in a field requiring a special date format, the client side part of the validation framework is trying to transform the input to a uniform representation by using a *correction validator*. If the conversion fails or the date is invalid because of another irregularity regarding validation rules, the validation presentation framework highlights the erroneous field and shows the user the defined validation error message. This process is additive, which means all validation errors are incrementally added to a client side *errors list* that can be shown on the user interface. Before submits the client side validation of all input fields within a triggered Validation Scope (cf. section 4.9) is done to avoid unnecessary roundtrips to the server. Because client side validation is not reliable, an identical validation on server side always takes place after submits. Validation processes on both the client and on the server side are completely transparent because the validation rules developer does only have to specify declarative validation rules (policies) and the mappings of the policies to the validation targets. Figure 14 shows an example of the additive and field based validation error visualization provided by our validation presentation framework. More details on our validation visualization approach framework will be published soon.

Fig. 14. Exemplary validation error visualization.

Regarding the validation setup, on application startup (or deployment) the validation Runtime Environment provided by the *AspNetWebValidationManager* executes a multi stage process (Figure 15) to create client and server side in-memory representations of the validation logic definitions and validation mapping definitions.

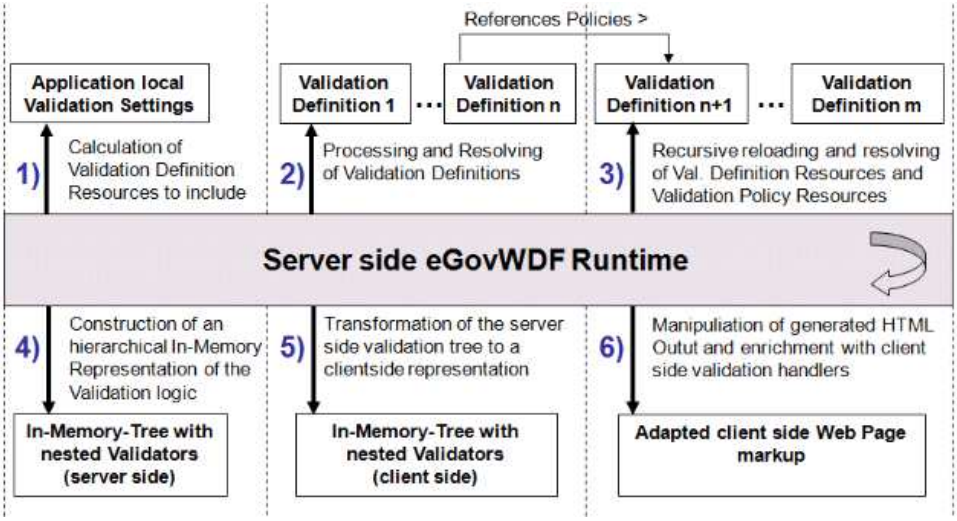

Fig. 15. Validation logic setup at application startup (or deployment).

As depicted in Figure 16, the In-Memory tree described above includes a flat structure of mappings between components and validation policies. Mappings formulated in the global context are processed on each *context* (Web page), whereas mappings of a local context are only applied if the corresponding local context is active, i.e. the corresponding Web page is currently processed.

In contrast to the mapping structure, the Policy tree which is also part of the validation logic in-memory representation can be arbitrary deep because validators and policies can be indefinitely nested.

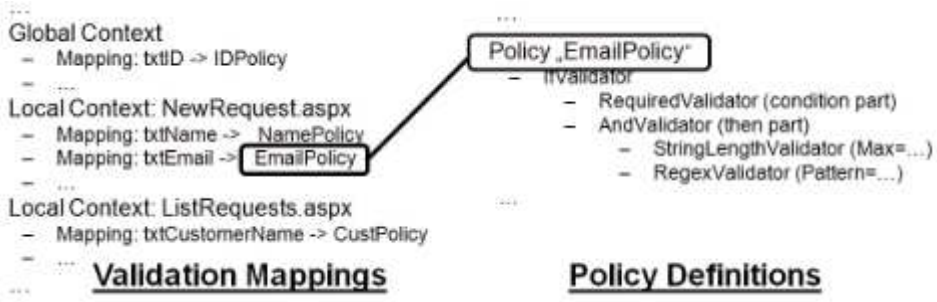

Fig. 16. In-Memory validation policy and validation mapping definitions tree.

4.9 Validation triggering

As mentioned above a *ValidationManager* connects an application and its components to an abstract definition of the validation logic which is described in policies. In this section we would like to describe the mechanisms of eGovWDF regarding validation triggering.

The eGovWDF approach implements the enactment of validation by the idea of *validation scopes* and *validation scope triggers*. A *validation scope* can be defined to include all components which are to be validated at the same time. This is similar to the *ValidationGroup* construct in ASP.NET. However, in eGovWDF validation scopes are defined purely declarative and can also be nested, merged and inherited. Another advantage of the validation scope approach is the support of multiple user interfaces, because the same validation scope definitions can be used for multiple user interfaces, e.g. a desktop and a web frontend.

The validation of a validation scope and in consequence of all its components is raised by so-called *validation triggers* which can be any components that are able to raise some event. Figure 17 and Listing 4 illustrate the validation scope concept and show that a component can be defined as trigger for an arbitrary number of validation scopes (e.g. button x is trigger of scope A and scope B) and a component can also be member of an arbitrary number of validation scopes (e.g. radio buttons r1 and r2 are both part of scope A and scope B). Components which are not explicitly associated to a validation scope are automatically added to a default scope (e.g. input field i and checkbox C2).

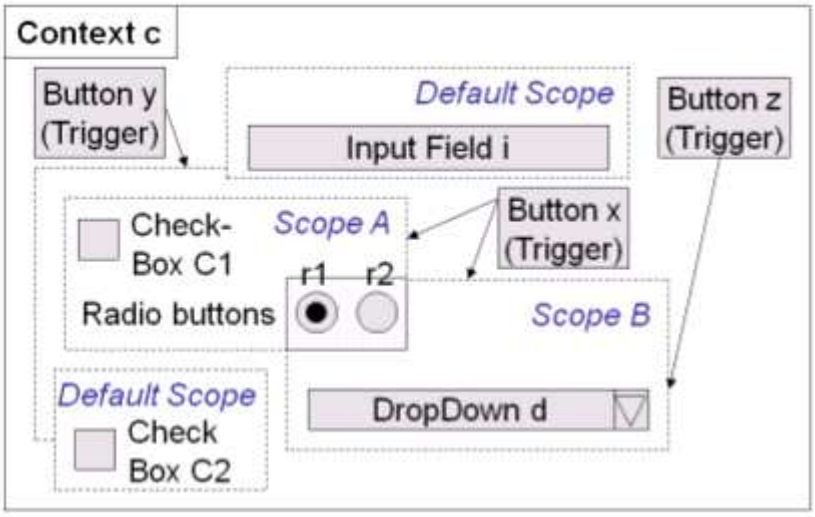

Fig. 17. Validation scopes, triggers and components.

```
<Context id="c">
  <ValidationMappings>
    <!-- Component and Policy mappings … -->
  </ValidationMappings>
```

```
<ValidationScopes>
 <Scope name="Scope A">
  <Triggers>
   <Trigger id="x"/><Trigger id="y"/>
  </Triggers>
  <Components>
   <Component id="C1" />
   <Component id="r1" /><Component id="r2"/>
  </Components>
 </Scope>
 <!-- Definition of Validation Scope B … -->
</ValidationScopes>
</Context>
```

Listing. 4. Validation Scope definitions in *ApplicationDefinitions XML format (ValADXF).*

By using validation scopes, validation mapping definitions and validation policies the validation related logic and behavior can be defined declaratively and *outside* of the application. This makes changes to the validation logic or behavior possible without the requirement for code changes. It also allows updates to the validation logic (mappings) at runtime and without the requirement for application redeployments and restarts.

5. EVALUATION OF EGOVWDF

Because our approach was designed as response to the non-availability [Kern 2008a] of a suitable solution for input validation in the Web context, our conception was strongly targeted at taking the postulated requirements (cf. section 3.2 and [Kern 2008a]) into appropriate account. This enabled a direct comparison between current solutions and our approach because it allowed us to contrast our framework with the same requirements the market analysis (cf. section 3) is based on.

In this section we show the performance of our approach regarding the requirements mentioned in section 3. We also compare our approach with the frameworks that were tested in the market analysis. A much more extensive comparison between our approach and the frameworks mentioned in section 3 is described in [Kern 2009].

5.1 Methodology

The evaluation methodology is almost identical to the one used in the market analysis. This means it was carried out in terms of an expert review and we used the same three levels of conformance to the requirements in order to make the results comparable:

- 0.0: No support of a requirement.
- 0.5: Partial support of a requirement.
- 1.0: Complete support of a requirement

However in addition to the calculation of maximum and average degree of requirement fulfillment, we also calculated mean value, lower quartile, median, upper quartile, minimum and maximum values of our approach and the frameworks of the market analysis to draw further conclusions. Additionally, we illustrated the data and the statistical characteristics by using bar charts and box plots [Tukey, 1977].

5.2 Execution

In this section we discuss the compliance level of eGovWDF to every single requirement which forms the basis for the statistical evaluation and discussion in section 5.3.

The separation of validation logic formulation, validation integration and validation error visualization is demanded as a minimum to make the implementation of these different aspects interchangeable (cf. Section 3.2). This aspect is considered by eGovWDF by the separation into various layers [Gamma et al. 1994] that are loosely coupled by interfaces and implemented as separate frameworks. All layers are built upon the Plugin Pattern [Fowler 2002]. As depicted in Figure 18, the framework consists of the following layers:

- *eGovWDF Validation RTE*: This is the *runtime environment* that processes validation logic and is executed on validation.
- *eGovWDF Validation PI*: The validation *platform integration* layer contains platform specific code to integrate validation processing in a specific platform. For each platform a *ValidationManager* (cf. Section 4.8) has to be provided. In the Web context, the *ValidationManager* setups a client side equivalent of the Validation RTE, installs client side event handlers on components to trigger validation and delegates validation processing to the Validation RTE.
- *eGovWDF Validation CL*: This layer contains the validation core logic which includes a basic set of validators such as RegexValidator, RequiredValidator,

IfValidator, ComponentReferenceValidator, PolicyReferenceValidator and Boolean composite validators.

- *eGovWDF Validation PF*: The presentation framework layer contains all components and functionality to visualize validation results.
- *eGovWDF Processors*: Processors provide core functionality used by all layers.

Figure. 18. Base architecture of the eGovWDF validation approach.

Because the requirement is completely fulfilled, the score for this requirement is 1.0.

Moreover, a declarative and rule based formulation [Ligeza 2006] of validation logic is requested. eGovWDF completely complies to this request because validation logic is defined in form of policies which are declarative specifications of nested validators. Additionally, validation mapping definitions are specified declarative, too. Complex validation rules can be implemented by nesting basic validators and including existing policies. This means a score of 1.0.

Furthermore, validation logic has to be reusable. As a consequence, concepts such as inheritance [Biddle and Tempero 1996], combinability and scalability [Bondi 2000; D'Antonio et al. 2004] are demanded. As mentioned in section 4.7, eGovWDF has substantial support for inheritance and scalability by its concept of *horizontal and vertical distribution* combined with the support of *inheritance options*. eGovWDF has also major support for combinability because by using compositional validators and validator nesting, complex validation scenarios can be implemented. This also includes conditional validation and inter-field dependencies and equals a score of 1.0.

In addition, validation logic is required to be extensible. Our approach supports this demand by a common *IValidator* interface that allows the implementation of arbitrary,

new validation logic. Furthermore, new validation logic can be created based on existing validation logic by inheriting from existing validators or referencing existing policies within a compositional validation policy definition.

Besides, plugin concepts for data storage (e.g. database, file system) and validation definition resource format are requested. eGovWDF supports both demands by a new approach to resource access called *Dimension Architecture*. The *Dimension Architecture* [Kern et al. 2010] is based on the separation of various aspects of resource access, e. g. location address, content format and type of data source into so-called *Dimensions* [Kern et al. 2010] which allows for the flexible and configurable combination of these aspects. As a result, eGovWDF achieves a score of 1.0 regarding this requirement.

Moreover, it is advocated that an approach should be downward compatible and support an unobtrusive as possible extension of legacy applications to find acceptance. eGovWDF is completely downward compatible because it is based on a principle we call *unobtrusive decoration*. This means eGovWDF does not require a specific entity or persistence framework to be used in Web applications. Instead, the application developer does only need to add a *ValidationManager* component (a server side user interface control) to the web page that requires validation. This component installs all required client side scripts and also triggers client side and server side validation based on the *user interface component events* specified in an application local configuration file. eGovWDF does not demand any modifications in the existing code data model or code; it only requires to add an additional *ValidationManager* component and to declaratively configure the validation behaviour. This also means a score of 1.0.

Beyond that, validation frameworks are demanded to support different validation severity levels. Our approach supports this requirement both at the validation logic level and at the user interface level. eGovWDF does not only support one additional state for a validation result such as *undetermined*, but also supports an arbitrary number validation severity levels. One can assign an arbitrary validation severity to any validator of a validation mapping definition. If a validator of a mapped policy is executed and evaluates to false, the assigned validation severity of the validator is returned which can be used to implement validation warning messages that do – in contrast to regular errors - not stop further processing, for instance. Our reference validation visualization framework also supports three severities out of the box. It contains a different visualization for information, warning and error messages. Because this part of our framework is also

based on the plugin pattern, further visualizations for additional severity levels can be implemented. This means a score of 1.0.

Finally, the requirements mentioned in section 3.2 include the demand for support of validation on both the client and on the server [Moore 2002] based on a declarative definition of validation logic and validation mappings. This is fully supported in eGovWDF because validation logic definitions and validation mapping definitions are declaratively described in *policy definition resources* and in *application definition resources*. As mentioned in section 4.8, these definitions are automatically translated into a client side and server side representation. Because the requirement is completely fulfilled, the score for this requirement is 1.0.

5.3 Discussion

The evaluation shows that eGovWDF completely fulfills the requirements postulated in section 3.2 of this paper. According to Kern [2009] it also fulfills all additional requirements stated in [Kern 2008a] such as tool support, Web accessibility and common requirements related to validation visualization. Because eGovWDF achieved the maximum degree of fulfillment in all requirements the overall score of eGovWDF equals the maximum achievable value. In numbers, the overall relative score of eGovWDF is 1,0 which equals 100 per cent of requirement fulfillment. The result is even more convincing when a comparison with other current Web input validation solutions is conducted because the best solution on the market only reaches a degree of fulfillment of 59 per cent (Figure 19).

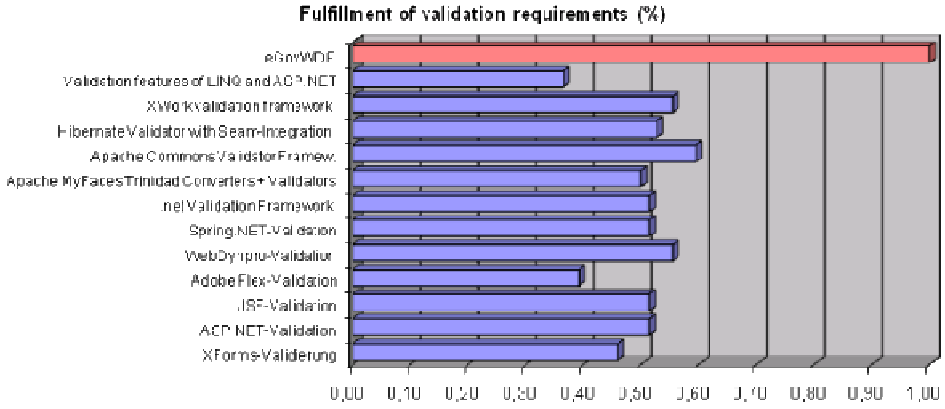

Figure. 19. Comparison of current validation frameworks regarding the fulfillment of main validation requirements in the eGovernment context.

In Figure 20, a box plot shows lower quartile, median, upper quartile, minimum and maximum values of the fulfillment of the validation requirements by eGovWDF and the frameworks mentioned in Kern [2008a]. The first box shows the distribution data without considering the fulfillment value of eGovWDF. The small median, lower quartile and upper quartile in both visualizations affirm the weak compliance of current validation frameworks mentioned above.

The short whisker lines combined with the small differences between median, lower quartile and higher quartile depicted in sub picture 1 of Figure 20 show that all frameworks are nearly at the same level of compliance that marks the state-of-the-art of input validation. This means there are no input validation frameworks that are significantly abreast of science. The data of the second box includes the degree of requirement fulfillment of eGovWDF. Median, lower and upper quartiles are nearly identical because of the small sensitivity of these statistical parameters to single values. However, the upper whisker ranges up to 1.00 which is the maximum degree of validation requirement compliance. This single value at the upper end of the upper whisker represents the degree of requirement compliance of eGovWDF and shows that eGovWDF significantly surpasses the state-of-the-art regarding input validation.

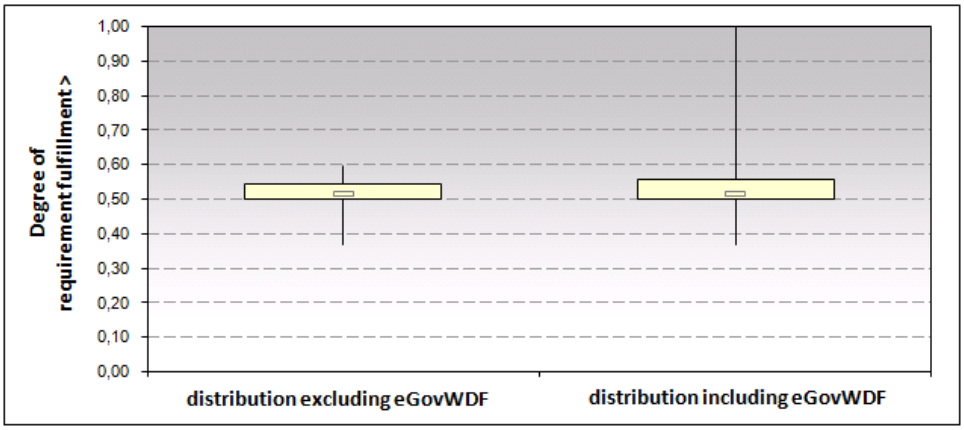

Figure. 20. Comparison of validation requirement distribution of current validation.

Further analysis [Kern 2009] shows that the differences between eGovWDF and other frameworks are most notable in the area of validation logic. As depicted in Figure 21, most validation frameworks only achieve between 30 and 40 per cent of compliance to

the validation logic related requirements. Even the best framework achieves only 50 per cent. Figure 22 details this situation by showing the compliance to the validation requirements for every single framework in one image. It shows that only eGovWDF (framework 13 in Figure 22) completely fulfills all requirements stated in Kern [2008a] and approves the weak compliance of all validation frameworks to validation logic related aspects.

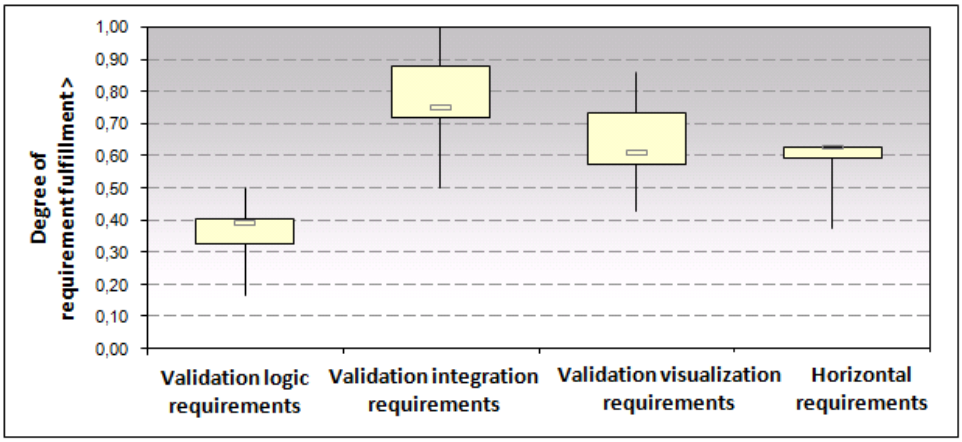

Figure. 21. Overview of requirement compliance regarding different validation areas.

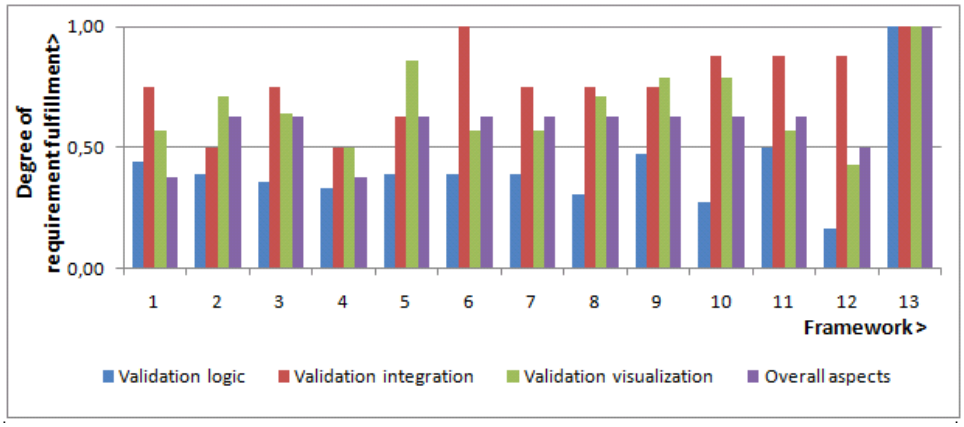

Figure. 22. Requirement compliance of current validation frameworks regarding different validation areas.

A more detailed analysis of current validation frameworks and eGovWDF can be found in [Kern 2008a] and in [Kern 2009].

Further steps will include the usage of eGovWDF in large scale projects like *BeihilfeOnline* (cf. section 1) which will be offered to all civil servants of Bavaria and

include about 450.000 users. After the implementation we will conduct at least three more studies:

- one to determine the usability of the framework for Web application developers,
- another one to ascertain the usability for non-IT-people to define validation logic and validation mappings
- and a third one to get feedback from the end users about the usability and accessibility of the eGovWDF based Web frontend.

6. RELATED WORK

In this section, we will discuss the most relevant scientific approaches on user input validation, but also address related topics such as *data type based validation* and *input validation testing* [Hayes and Offutt 1998; Li et al. 2007]. More details on the advantages and shortcomings of current (commercial) validation frameworks are mentioned in section 5 and are examined in [Kern 2008a].

6.1 PowerForms

PowerForms is a high-level domain-specific language that supports incremental input validation on the client side [Brabrand et al. 2000]. It allows the declarative specification of valid formats for HTML input components based on regular expressions. Inter-field dependencies are supported by special tags (<*if*>, <*then*>, <*else*>) that can be used within the format specification of a component.

Compared to eGovWDF, PowerForms has several shortcomings. PowerForms depends heavily on client side script because all declarative validation format specifications are transformed to client script, but not to server side code. Because the end users browser may not support client side scripting and client side code can be manipulated, server side validation is essential. As a consequence the developer has to manually implement all PowerForms generated client side code as server side code.

Additionally, PowerForms is based on regular expressions and a domain specific language with a fixed set of features. Although regular expressions are a powerful tool, there may be situations when regular expression based validation is not possible [Scaffidi et al. 2008] or more efficient alternatives are available. For that reason, eGovWDF is based on an extensible plugin architecture that supports not only regular expressions but arbitrary validation logic in the form of different validator types, e.g. *CurrencyValidator*.

Both, PowerForms and eGovWDF support conditional and logical operators, but in PowerForms this operators are language intrinsic. In eGovWDF, all validation logic related things - including conditional expressions, composition and dependencies - are implemented as replaceable and extendable validators. In contrast to PowerForms, eGovWDF does not explicitly support Java applets because our frameworks targets pure Web applications. This decision is motivated by the fact that Java applets require an installed plugin in the clients' browser which could have negative effects on Web accessibility [W3C 1999].

6.2 The *Topes approach*

Scaffidi et al. describe an approach that is based on *topes* [Scaffidi et al. 2008]. A *tope* is an "application-independent abstraction describing how to recognize and transform values in a category of data" [Scaffidi et al. 2008]. Examples for topes are email addresses, company names or amounts of money. The abstraction represented by a tope can be described by several patterns, e.g. to describe a company, patterns for all kinds of company representations (stock symbol notation, company abbreviation …) can be developed and implemented. This means, topes support the specification of alternative representations for data, e.g. the company *Microsoft* can be represented by *Microsoft Corporation*, *Microsoft*, *MS* or *MSFT*. Moreover this approach supports reuse by extending the general purpose formats of topes with specialized formats on demand.

Compared to our approach, a tope has similarities with a validation policy. Both a tope and a policy contain reusable validation logic. The equivalents of tope patterns are correction validators. However, because our approach aims in providing a comprehensive solution to input validation, some of the more advanced features of our approach have no counterpart in the topes approach. Our approach is based on a detailed inheritance and distribution model (cf. section 4.7), has an extensive support for component and validation logic dependencies and allows the flexible combination of existing validation logic modules in policies. Additionally, topics such as triggering of actions as result of validation processes and conditional validation are also supported in eGovWDF. One possible advantage of the tope approach is its support for non-binary validation. This means the tope approach supports the classification of input to be neither valid nor invalid. The *third state* can be used to give the user warnings about his input but does not have a rejection as consequence. However, this feature can be emulated in eGovWDF by using its support of validation severities. eGovWDF does not only support one additional

state for a validation result such as *undetermined*, but also supports an arbitrary number validation severity levels. One can assign an arbitrary validation severity to any validator of a validation mapping definition. If a validator of a mapped policy is executed and evaluates to false (in case the input does not match a *pattern*), the assigned validation severity of the validator is returned which can be used to implement validation warning messages that do – in contrast to regular errors - not stop further processing, for instance. The topes approach also supports tool based definition of topes for end user programmers which is similar to our approach. Our approach targets non-IT people that specify the functional validation requirements.

Summing up, both eGovWDF and the topes approach support input validation and multiple representations of input data. However, eGovWDF provides a more comprehensive approach than the *topes solution* because the topes approach is more specialized and probably better suited in the area of data abstractions.

6.3 Other input validation approaches

Prior to the development of our approach to input validation, we conducted a study [Kern 2008a] on the fulfillment of current Web validation frameworks regarding the validation logic related requirements mentioned in section 3.2 and additional requirements such as tool support, Web accessibility and validation visualization. The evaluation showed only 51 percentages of average fulfillment of the requirements and 59 per cent maximum fulfillment by *Apache Commons Validator framework*. Moreover, our evaluation showed that most validation frameworks, including Apache Commons Validator, fail in the requirements that are most important to a flexible validation approach such as reusability (none of the frameworks achieved more than 50 per cent fulfillment of this requirement).

6.4 Further related work

Data types [Cardelli and Wegner 1985] describe the kind and structure of data which includes the specification of the range of valid values. As mentioned before, Web application input components could possibly hold any value [Brabrand et al. 2000] because there is no support for data type specification or even stricter constraints. However, some Web application frameworks support the mapping of HTML input fields to strongly typed instance members of a server side object. Such bindings help the server side framework to check user input against data type constraints on submits. Some frameworks, such as JSF [Burns and Kitain 2006], additionally support converter

components that convert the user supplied string data into the mapped data types in case simple conversion is not possible. Even though, *data type based validation is too coarse-grained for even simple validation*, e.g. validation of data that is to be matching a certain regular expression.

Another relevant topic in the domain of *input validation* is *input validation testing (IVT)* [Hayes and Offutt 1998; Li et al. 2007]. According to Offutt and Hayes, input validation testing is defined as "choosing test data that attempt to show the presence or absence of specific faults pertaining to input-tolerance" [Hayes and Offutt 1998]. In this context, input-tolerance means the ability of applications to handle expected and unexpected user input. Because input validation testing aims in testing the application behavior in case of unexpected inputs, IVT is a suitable solution to check the implementation of input validation. Regarding eGovWDF, IVT approaches can be used to check single validators or complex scenarios by assigning the validators to be tested to user interface components that are filled with IVT test data.

Furthermore, input validation can be seen as a kind of exception handling. Although exception handling originates from the field of programming languages, this concept can be adapted to other fields, too [Kienzle 2008]. By definition, an exception describes a situation that – if occurring (during program execution) - requires an extraordinary reaction to resolve it. There are several approaches which apply exception handling to the complete software lifecycle [Kienzle 2008]. Most of them advocate a separation of exception handling related aspects [Lemos and Romanovsky 2001; Lippert and Lopes 2000; Filho et al., 2006] which is also the idea of our approach. However our approach takes this idea of separation of aspects even further by implementing lose coupling on different levels ob abstraction (validators, mappings, policies). In addition, in contrast to software development related exception handling approaches, our approach directly targets the input validation domain and as a consequence is generally better suited for requirements in this domain. The same argumentation applies for defensive programming [Gilmour 1990; Miller et al. 2009] which aims at creating reliable computer programs based on secure programming practices.

Finally, there are also similarities between input validation and constraint validation. According to Froihofer et al. [2007], constraint validation is one of the most essential tasks of a system to ensure integrity which is an important attribute of reliability and security. Similar to the input validation domain, there are competitive approaches, ranging from simple and hard-coded if-then-clauses to more flexible and declarative

approaches. Classic if-then-based approaches are easy to implement, but they are very unflexible and do not allow a specification on higher levels of abstractions, e.g. a visual specification by non-IT people that usually formulate requirements. In addition, this paradigm leads to development costs if changes in the business logic are required. On the other hand, code instrumentation based approaches and compiler based approaches allow the separation of these aspects and glue the aspects together at runtime or compile time. However these approaches also have major disadvantages regarding code duplication, debugging problems and compatibility problems [Froihofer et al., 2007]. As a consequence, approaches that feature explicit constraint classes and interceptor based approaches are recommended if flexibility is required and performance is of secondary relevance [Froihofer et al., 2007] which reinforces our approach because of its separation of different types of validation logic at various levels of abstraction.

7. CONCLUSIONS AND FUTURE WORK

This paper presented the validation logic related aspects of a novel approach to input validation in the Web context. Our approach supports the declarative, rule based definition of validation logic and mappings between validation logic and the components to validate. eGovWDF has extensive support for combination, extension and inheritance of validation logic. It considers validation as comprehensive process which takes value checking as well as data normalization and user guidance into account. It provides a uniform architecture for all types of validation. For instance, conditional validation, value verification, value correction, dependencies and validation result depending actions are all implemented as validators.

Recently, we conducted a study [Kern 2009] that compared our approach to the requirements mentioned in section 3.2 and in [Kern 2008a]. The evaluation approved our work showing 100 per cent compliance to the requirements. Currently, we are refining our graphical designer tool which provides a WYSIWYG support for validation logic definitions and validation mappings definitions. This tool, which is targeted at non-IT people that usually provide functional validation requirements, also includes a validation logic test center to simulate validation logic definitions even before they are used in Web applications. In this context we consider to introduce a new role in software development called “validation engineer” which correlates to the “exception engineer” proposed by [Shah et al., 2008]; however in contrast to the exception engineer our validation engineer is a non-IT person.

Further steps will be the usage in large scale projects of the Government like *BeihilfeOnline* (cf. section 1) with about 480.000 users. After the implementation we plan to do some studies – one to determine the usability of the framework for Web application developers, another one to ascertain the usability for non-IT-people to define validation logic and validation mappings and a third one to get feedback from the end users about the usability and accessibility of the eGovWDF based Web frontend.

REFERENCES

ALJAWARNEH, SHADI, LAING, CHRISTOPHER, AND VICKERS, PAUL. 2007. Verification of Web Content Integrity: Formulating models to protect servers against tampering. In: MERABTI, M (ed), PGNET 2007 The 8th Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting. Liverpool John Moores University.

AMBER, SCOTT. 1996. Object-relational mapping. Softw. Dev., 4(10), 47–50.

BALMIN, ANDREY, PAPAKONSTANTINOU, YANNIS, AND VIANU, VICTOR. 2004. Incremental validation of XML documents. vol. 29. New York, NY, USA: ACM.

BIDDLE, ROBERT, AND TEMPERO, EWAN. 1996. Explaining inheritance: a code reusability perspective. Pages 217–221 of: SIGCSE '96: Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education. New York, NY, USA: ACM.

BONDI, ANDRÉ B. 2000. Characteristics of scalability and their impact on performance. Pages 195–203 of: WOSP '00: Proceedings of the 2nd international workshop on Software and performance. New York, NY, USA: ACM.

BORTZ, ANDREW, AND BONEH, DAN. 2007. Exposing private information by timing web applications. Pages 621–628 of: WWW '07: Proceedings of the 16th international conference on World Wide Web. New York, NY, USA: ACM.

BOUKERCHE, AZZEDINE, AND TROPPER, CARL. 1998. A Distributed Graph Algorithm for the Detection of Local Cycles and Knots. IEEE Trans. Parallel Distrib. Syst., 9(8), 748–757.

BRABRAND, CLAUS, MØLLER, ANDERS, RICKY, MIKKEL, AND SCHWARTZBACH, MICHAEL I. 2000. PowerForms: Declarative Client-Side Form Field Validation. World Wide Web Journal, 3, 2000.

BRINHOSA, RAFAEL BOSSE, WESTPHALL, CARLOS BECKER, AND WESTPHALL, CARLA MERKLE. 2008. A Security Framework for Input Validation. Pages 88–92 of: SECURWARE '08: Proceedings of the 2008 Second International Conference on Emerging Security Information, Systems and Technologies. Washington, DC, USA: IEEE Computer Society.

BROY, MANFRED, DEISSENBOECK, FLORIAN, AND PIZKA, MARKUS. 2006. Demystifying maintainability. Pages 21–26 of: WoSQ '06: Proceedings of the 2006 international workshop on Software quality. New York, NY, USA: ACM.

BUEHRER, GREGORY, WEIDE, BRUCE W., AND SIVILOTTI, PAOLO A. G. 2005. Using parse tree validation to prevent SQL injection attacks. Pages 106–113 of: SEM '05: Proceedings of the 5th international workshop on Software engineering and middleware. New York, NY, USA: ACM.

BURNS AND KITAIN (ed) 2006. JavaServer Faces Specification: Version 1.2 - Rev A.

BUSCHMANN, FRANK, MEUNIER, REGINE, ROHNERT, HANS, SOMMERLAD, PETER, AND STAL, MICHAEL. 1996. Pattern-Oriented Software Architecture Volume 1: A System of Patterns. 1 edn. Wiley.

CARDELLI, LUCA, AND WEGNER, PETER. 1985. On Understanding Types, Data Abstraction, and Polymorphism. ACM Computing Surveys, 17, 471–522.

CARDONE, RICHARD, SOROKER, DANNY, AND TIWARI, ALPANA. 2005. Using XForms to Simplify Web Programming.

CHENG, JINGWEN. 1994. A reusability-based software development environment. SIGSOFT Softw. Eng. Notes, 19(2), 57–62.

CHRISTENSEN, ASKE SIMON, MOELLER, ANDERS, AND SCHWARTZBACH, MICHAEL I. 2003. Precise Analysis of String Expressions. Pages 1–18 of: SAS '03: Proceedings of the 10th International Static Analysis Symposium. Springer-Verlag.

CRISTEA, A.D., AND PROSTEAN, O. 2009. Dynamic programming with Web Dynpro ABAP, Pages 173–176 of: SACI '09: 5th International Symposium on Applied Computational Intelligence and Informatics. IEEE.

D'ANTONIO, S., ESPOSITO, M., ROMANO, S. P., AND VENTRE, G. 2004. Assessing the scalability of component-based frameworks: the CADENUS case study. SIGMETRICS Perform. Eval. Rev., 32(3), 34–43.

DE LEMOS, R., AND ROMANOVSKY, A. 2001. Exception Handling in the Software Lifecycle. International Journal of Computer Systems Science and Engineering. 16(2), 119–133.

DE PAULA, MAIRA GRECO, DA SILVA, BRUNO SANTANA, BARBOSA, SIMONE DINIZ JUNQUEIRA. 2005. Using an interaction model as a resource for communication in design. Pages 1713–1716 of: CHI '05: CHI '05 extended abstracts on Human factors in computing systems. New York, NY, USA: ACM.

DEUTSCES BUNDESMINISTERIUM DES INNERN (ed) 2002. Barrierefreie Informationstechnik-Verordnung (BITV).

ELLUL, KEITH, KRAWETZ, BRYAN, SHALLIT, JEFFREY, AND WANG, MING-WEI. 2005. Regular expressions: new results and open problems. J. Autom. Lang. Comb., 10(4), 407–437.

FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., AND BERNERS-LEE, T. 1997. Hypertext Transfer Protocol – HTTP/1.1.

FILHO, F. C., CACHO, N., FIGUEIREDO, E., MARANHAO, R., GARCIA, A. AND RUBIRA, C. M. F. 2006. Exceptions and aspects: the devil is in the details. Pages 152–162 of: SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering. New York, NY, USA: ACM.

Forrester Research Inc. 2008. Forrester: Global Enterprise Web 2.0 Market To Reach $4.6 Billion By 2013. Last accessed at 14.06.2008.

FOWLER, MARTIN. 2002. Patterns of Enterprise Application Architecture (Addison-Wesley Signature Series). Addison-Wesley Professional.

FROIHOFER, LORENZ, GLOS, GERHARD, OSRAEL, JOHANNES AND GOESCHKA, KARL M. 2007. Overview and Evaluation of Constraint Validation Approaches in Java. Pages 313–322 of: ICSE '07: Proceedings of the 29th international conference on Software Engineering. Washington, DC, USA: IEEE Computer Society.

GAMMA, ERICH, HELM, RICHARD, JOHNSON, RALPH, AND VLISSIDES, JOHN. 1994. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, Massachusetts: Addison Wesley.

GAO, QIANG, GROZ, R., VON BOCHMANN, G., DARGHAM, J., AND HTITE, E. H. 1995. Validation of distributed algorithms and protocols. Page 110 of: ICNP '95: Proceedings of the 1995 International Conference on Network Protocols. Washington, DC, USA: IEEE Computer Society.

GARRETT, J. J. 2005. Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/ideas/essays/archives/000385.php. - Last accessed at 11.10.2008.

GILMOUR, P. S. 1990. Defensive programming. Embedded Syst. Program. 3(4), 60–68, San Francisco, CA, USA: Miller Freeman Inc.

GOLLMANN, DIETER. 2008. Securing Web applications. Inf. Secur. Tech. Rep., 13(1), 1–9.

GAO, QIANG, GROZ, R., VON BOCHMANN, G., DARGHAM, J., AND HTITE, E. H. 1995. Validation of distributed algorithms and protocols. Page 110 of: ICNP '95: Proceedings of the 1995 International Conference on Network Protocols. Washington, DC, USA: IEEE Computer Society.

GRAUNKE, PAUL, FINDLER, ROBERT BRUCE, KRISHNAMURTHI, SHRIRAM, AND FELLEISEN, MATTHIAS. 2003. Modeling web interactions. Pages 238–252 of: ESOP'03: Proceedings of the 12th European conference on Programming. Berlin/Heidelberg: Springer-Verlag.

HAYES, JANE HUFFMAN, AND OFFUTT, A. JEFFERSON. 1999. Increased Software Reliability Through Input Validation Analysis and Testing. Page 199 of: ISSRE '99: Proceedings of the 10th International Symposium on Software Reliability Engineering. Washington, DC, USA: IEEE Computer Society.

HEWITT, T., BAECKER, R., CARD, S., CAREY, T., GASEN, J., MANTEI, M., PERLMAN, G., STRONG, G., AND VERPLANK, W. 1992. ACM SIGCHI Curricula for Human-Computer Interaction.

JACKSON, COLLIN, BORTZ, ANDREW, BONEH, DAN, AND MITCHELL, JOHN C. 2006. Protecting browser state from web privacy attacks. Pages 737–744 of: WWW '06: Proceedings of the 15th international conference on World Wide Web. New York, NY, USA: ACM.

JAUVIN, NICOLE. 2007. Demographic challenges facing the federal public sector. http://www.tbs-sct.gc.ca/nou/n20070417-eng.asp. - Last accessed at 16.05.2009.

JIM, TREVOR, SWAMY, NIKHIL, AND HICKS, MICHAEL. 2007. Defeating script injection attacks with browser-enforced embedded policies. Pages 601–610 of:

WWW '07: Proceedings of the 16th international conference on World Wide Web. New York, NY, USA: ACM.

KAMIL, AMIR. 2003. Discussion 13: Topics: Graph Algorithms. http://www.cs.berkeley.edu/~kamil/teaching/fa02/120502.pdf. - Last accessed at 14.12.2008.

KBST. (ed) 2006. V-Modell XT 1.2.1. ftp://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/1.2.1/Dokumentation/V-Modell-XT-Gesamt.pdf. - Last accessed at 22.05.2009

KBST. (ed) 2008 (März). SAGA 4.0: Standards und Architekturen für E-Government-Anwendungen.

KERN, WALTER. 2008a. Validierungsunterstützung in Frameworks zur Webentwicklung: Eine Evaluierung der bedeutendsten Webentwicklungsframeworks im Hinblick auf den Aspekt Validierung im Kontext von eGovernment. Vdm Verlag Dr. Müller.

KERN, WALTER. 2008b. Web 2.0 - End of Accessibility? Analysis of Most Common Problems with Web 2.0 Based Applications Regarding Web Accessibility. International Journal of Public Information Systems, 02(2), 131–154.

KERN WALTER. 2010. Evaluierung des Webentwicklungsframeworks eGovWDF im Hinblick auf den Aspekt Validierung und die besonderen Anforderungen im Bereich des eGovernment. Technical report, Information Science Department of the University of Regensburg. http://epub.uni-regensburg.de/15763/. – Last accessed at 14.07.2010

KERN, WALTER, SILBERBAUER, CHRISTIAN and WOLFF, CHRISTIAN. 2010. "The Dimension Architecture: A New Approach to Resource Access," IEEE Software, pp. 74-81, September/October, 2010

KIENZLE, JÖRG. 2008. On exceptions and the software development life cycle. Pages 32–38 of: WEH '08: Proceedings of the 4th international workshop on Exception handling. New York, NY, USA: ACM.

LAPRIE, JEAN-CLAUDE, R, BRIAN, AND L, CARL. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, 1, 11–33.

LAWTON, G. 2008. New Ways to Build Rich Internet Applications. In: Computer 41 (2008), 10–12

LEE, W. 1987. ?: a context-sensitive help system based on hypertext. Pages 429–435 of: DAC '87: Proceedings of the 24th ACM/IEEE conference on Design automation. New York, NY, USA: ACM.

LI, NUO, ZHONG JIN, MAO, AND LIU, CHAO. 2007. Web Application Model Recovery for User Input Validation Testing. Page 13 of: ICSEA '07: Proceedings of the International Conference on Software Engineering Advances (ICSEA 2007). Washington, DC, USA: IEEE Computer Society.

LIGEZA, ANTONI. 2006. Logical Foundations for Rule-Based Systems. (Studies in Computational Intelligence). 2nd ed. edn. Springer, Berlin.

LIPPERT, M., AND LOPES, C. V. 2000. A study on exception detection and handling using aspect-oriented programming. Pages 418–427 of: ICSE '00: Proceedings of the 22nd international conference on Software engineering. New York, NY, USA: ACM.

LIU, HUI, AND TAN, HEE BENG KUAN. 2006. Automated verification and test case generation for input validation. Pages 29–35 of: AST '06: Proceedings of the 2006 international workshop on Automation of software test. New York, NY, USA: ACM.

LIU, HUI, AND TAN, HEE BENG KUAN. 2008. An approach for the maintenance of input validation. Inf. Softw. Technol., 50(5), 449–461.

MAHMOOD, ZAIGHAM. 2007. Service oriented architecture: potential benefits and challenges. Pages 497–501 of: ICCOMP'07: Proceedings of the 11th WSEAS International Conference on Computers. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS).

MICROSOFT CORPORATION (ed). 2003. Improving Web Application Security: Threats and Countermeasures. Microsoft Press Corp. Chap. 4, page 74.

MICROSOFT CORPORATION (ed). 2007. .NET Framework-Klassenbibliothek: BaseValidator.ValidationGroup-Eigenschaft. http://msdn.microsoft.com/de-de/library/system.web.ui.webcontrols.basevalidator.validationgroup(printer).aspx. - Last accessed at 23.05.2009.

MILLER, FREDERIC P., VANDOME, AGNES F., AND MCBREWSTER, J. 2009. Defensive Programming. Alphascript Publishing.

MILLER, ROBERT C., AND MYERS, BRAD A. 2001. Outlier Finding: Focusing User Attention on Possible Errors.

MITRE CORPORATION 2007. 2009 CWE/SANS Top 25 Most Dangerous Programming Errors. http://cwe.mitre.org/top25/#CWE-20. - Last accessed at 22.05.2009.

MONTGOMERY, MIKE, AND DANIEL, DWIGHT D. 2009. End user developer: friend or foe? J. Comput. Small Coll., 24(4), 40–45.

MOORE, ANTHONY. 2002. ASP.NET Validation in Depth. http://msdn2.microsoft.com/en-us/library/aa479045(printer).aspx. - Last accessed at 09.01.2008.

NANCE, RICHARD E., AND ARTHUR, JAMES D. 1994. Software Quality Measurement: Assessment, Prediction and Validation. Tech. rept. Blacksburg, VA, USA.

NIELSEN, JAKOB. 2005. Jakob Nielsen's Alertbox, September 19, 2005: Forms vs. Applications. http://www.useit.com/alertbox/forms.html. - Last accessed at 12.01.2008.

NIERSTRASZ, GIBBS, AND TSICHRITZIS. 1992. Component-oriented software development. Commun. ACM, 35(9), 160–165.

NIVASCH, GABRIEL. 2004. Cycle detection using a stack. Inf. Process. Lett., 90(3), 135–140.

OFFUTT, JEFF, WU, YE, DU, XIAOCHEN, AND HUANG, HONG. 2004. Bypass Testing of Web Applications. Pages 187–197 of: ISSRE '04: Proceedings of the 15th International Symposium on Software Reliability Engineering. Washington, DC, USA: IEEE Computer Society.

OLSEN, KAI A. 2008. The $100,000 Keying Error. Computer, 04, 106–108.

O'REILLY, TIM. 2005. What Is Web 2.0? Design Patterns and Business Models for the Next Generation of Software. Last accessed at 14.06.2008.

PACE, DALE K. 2003. Verification, validation, and accreditation of simulation models. 487–506.

PAULSON, L. D. 2005. Building Rich Web Applications with Ajax, Computer, v. 38 n. 10, p. 14-17.

PRESSMAN, ROGER S. 1996. Software Engineering: A Practitioner's Approach. 5. edn. McGraw-Hill Higher Education.

SARGENT, ROBERT G. 1987. An overview of verification and validation of simulation models. Pages 33–39 of: WSC '87: Proceedings of the 19th conference on Winter simulation. New York, NY, USA: ACM.

SCAFFIDI, CHRISTOPHER, MYERS, BRAD, AND SHAW, MARY. 2008. Topes: reusable abstractions for validating data. Pages 1–10 of: ICSE '08: Proceedings of the 30th international conference on Software engineering. New York, NY, USA: ACM.

SCHALL, ERIC. 1990. Parallel cycle detection in distributed databases. Inf. Syst., 15(5), 555–566.

SCHLUMBERGER, M. 1989. Definition and validation of user interfaces. Pages 518–525 of: Proceedings of the third international conference on human-computer interaction on Designing and using human-computer interfaces and knowledge based systems (2nd ed.). New York, NY, USA: Elsevier Science Inc.

SEDAYAO, JEFF. 2008. Implementing and operating an internet scale distributed application using service oriented architecture principles and cloud computing infrastructure. Pages 417–421 of: iiWAS '08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services. New York, NY, USA: ACM.

SIVATHANU, GOPALAN, WRIGHT, CHARLES P., AND ZADOK, EREZ. 2005. Ensuring data integrity in storage: techniques and applications. Pages 26–36 of: StorageSS '05: Proceedings of the 2005 ACM workshop on Storage security and survivability. New York, NY, USA: ACM.

SHAH, HINA, GÖRG, CARSTEN, AND HARROLD, MARY JEAN. 2008. Why do developers neglect exception handling?. Pages 62–68 of: WEH '08: Proceedings of the 4th international workshop on Exception handling. Atlanta, Georgia: ACM.

Taivalsaari, A. 1996. On the notion of inheritance. *ACM Comput. Surv.* 28(3):438-479.

TERAGUCHI, MASAYOSHI, YOSHIDA, ISSEI, AND URAMOTO, NAOHIKO. 2008. Rule-based XML Mediation for Data Validation and Privacy Anonymization. Pages

21–28 of: SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing. Washington, DC, USA: IEEE Computer Society.

TUKEY, JOHN W. 1977. Exploratory Data Analysis. Addison Wesley Pub Co Inc.

W3C (ed) 1999. Web Content Accessibility Guidelines 1.0. http://www.w3.org/TR/WAI-WEBCONTENT/. - Last accessed at 14.06.2008.

W3C (ed) 2001. XForms Requirements: W3C Working Draft 04 April 2001. http://www.w3.org/TR/xhtml-forms-req.html. - Last accessed at 21.05.2009.

W3C (ed) 2005. Introduction to Web accessibility. http://www.w3.org/TR/WAI-WEBCONTENT/. - Last accessed at 20.05.2009.

W3C (ed) 2008a. Web Content Accessibility Guidelines 2.0. http://www.w3.org/TR/WCAG20/ - Last accessed at 12.01.2008.

W3C (ed) 2008b. Understanding WCAG 2.0. http://www.w3.org/TR/UNDERSTANDING-WCAG20/complete.html. - Last accessed at 18.05.2009.

WANG, FEI. 2009. The Development of Rich Internet Application Based on Current Technologies. Pages 815–818 of: WISM '09: Proceedings of the 2009 International Conference on Web Information Systems and Mining.

WASSERMAN, G., AND SU, Z. 2004. An analysis framework for security in web applications. Pages 70–78 of: SAVCBS '04: Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems.

WEINREICH, HARALD, OBENDORF, HARTMUT, HERDER, EELCO, AND MAYER, MATTHIAS. 2008. Not quite the average: An empirical study of Web use. ACM Trans. Web, 2(1), 1–31.

WEST, C. H. 1989. Protocol validation in complex systems. Pages 303–312 of: SIGCOMM '89: Symposium proceedings on Communications architectures & protocols. New York, NY, USA: ACM.

WILLIAMS, JEFF. 2004. Input Validation. AppSec 2004. http://www.owasp.org/images/3/37/ AppSec2004-Jeff_Williams-Input_Validation.ppt. - Last access at 01.02.2009.

YANG, FAN, GUPTA, NITIN, GERNER, NICHOLAS, QI, XIN, DEMERS, ALAN, GEHRKE, JOHANNES, AND SHANMUGASUNDARAM, JAYAVEL. 2007. A unified platform for data driven web applications with automatic client-server partitioning. Pages 341–350 of: WWW '07: Proceedings of the 16th international conference on World Wide Web. New York, NY, USA: ACM.

YUE, CHUAN, AND WANG, HAINING. 2009. Characterizing insecure javascript practices on the web. Pages 961–970 of: WWW '09: Proceedings of the 18th international conference on World wide web. New York, NY, USA: ACM.

ZHANG, XIAO, ZHANG, YI, AND WU, JUN. 2007. Research and Analysis of Ajax Technology Effect on Information System Operating Efficiency. Pages 641–649 of: CONFENIS (1).