Web Information Systems as Abstract State Services

WISM’08
Barcelona
22. 10. 2008

Hui Ma, Klaus-Dieter Schewe, Bernhard Thalheim, Qing Wang

Victoria, Kiel Universities, New Zealand & Germany
Information Science Research Centre, Palmerston North, New Zealand
Overview

- Why
  Challenges, deficiencies, opportunities, hopes
- WIS ?!?
  Web information systems
  web services for information intensive systems
- Foundations
  Semantic foundations, reasoning, validation, verification, proofs, evolution
- States
  Appropriate depending on the user, component
- Services
  Towards a modern theory of web services
- Personalisation
  One of the main challenges nowadays
Background and Motivation

Specification framework for flexible web services
Theoretical foundation of web services
Reasoning framework
Mapping facilities for infrastructure, database, network etc. support

Challenges

Contextualisation as the biggest task nowadays

Personalisation as the ultimate support for any system

Specification for a flexible, configurable (CMDB), high-quality web system

Integration on actual demand and context

Foundation and Validation, Verification, Proof

What we did achieve so far?

Storyboarding for web information systems
Media types as the main mediating structures between a user and a web information system
Our Goals

Starting Points

- Take an abstract, conceptual approach to service integration, composition and personalisation

- A (data-intensive) service can be described by two layers:
  - a hidden database layer consisting of a database schema and transactions
  - a visible view layer on top of it providing views and functions based on them

- This idea appears for dialogue systems, Web Information Systems, component-based systems
The Idea of Abstract State Services

- Define a concept of abstract services for modern web information systems
- Develop a theory of Abstract State Services (ASSs) following the line of thought of the ASM thesis (Blass, Gurevich)
  - Formalise sequential and parallel algorithms by a small set of intuitive, abstract postulates
  - Prove that these postulates are always satisfiable by (sequential) ASMs
- Carry this idea over to database transformations (queries, updates)
- This forms the basis of the formal definition of ASSs by means of postulates
Service Integration, Composition, Personalisation

- Integration means to replace two or more ASSs by a single new one that offers all the functionality of the individual services.

- Reduced integration to database schema and view integration.

- Composition requires the extraction of service components from existing ASSs that feed a new service without replacing the original ones.

- We also need composition operations.

- Personalisation directs the extraction of suitable components by preferences.
Web Information Systems

From Simple Web to Web Information Systems

- Web Information System (WIS) is a database-backed information system that is realised and distributed over the web with user access via web browsers.
- A WIS is open to (almost) every user, but access in a particular role may be restricted.
- Websites have gone through three stages: Web 1.0, Web 2.0, Web 3.0.
  - Web 1.0 is mainly author-driven.
  - Web 2.0 is user driven and content centered.
  - Web 3.0 is characterised by (4C + P + VS).
Web 1.0

Our past, present and also future: Web 1.0 has mainly been oriented towards content provision to deliver content with a rudimentary functionality, e.g., navigation, acquisition information, linking, search and browse.

Achievements of Web 1.0 e.g.

- Resulting websites are simple to use, without any learning effort
- Based on an application development according to application scenarios
- It models story spaces as schemes for utilization
- It uses security techniques provided by server-sided logic and aiming being robust
- It uses taxonomies for classification and systematisation

Limitations of Web 1.0 e.g.

- It is often resulted in high time exposure for stepwise page buildup
- It is based on fixed page content
- It uses information dissemination from a central source
- It pushes information to users
- It is based on a unidirectional communication
- It does not providing relation to desktop organization
Example 1: Wiki Storyboards

What we did achieve already!!

forming a Wiki team with three roles (evaluator, member, supporter)
modelling the Wiki collaboration story
modelling the intended result
Example 2: Actors in a Paper Submission and Reviewing System

PC (Co-)Chair: via chair login
- list of overviews, all data, abstracts and papers
- paper assignment, deblock of data, PC decisions

PC Member: via personal member login with abstract/paper download
- before assignments: abstracts of submitted papers
- entire own review
- before PC session, after completing reviews: concurrent reviews
  - conflicts with concurrent reviews, discrepancies
- during PC session: anonymous survey reviews
- before assignments: indicate levels of interest
- until PC session: input / modify reviews for assigned papers
- until PC session: obtain / submit review template for assigned papers

Author: via entry page or personal author login
- general and his data
- before submission deadline: submit abstract / paper
- after PC session: obtain anonymous reviews, submit final version

Administrator: via admin tools and direct access (encrypted data)
- maintenance of software code, database, password update
Example 2: Data Structuring for a Paper Submission and Reviewing System

PCPhases: CMEditProfile; CBrowseAbstr; ReviewsDue; CMViewOther; Session
AuPhases: ASubmAbstr; ANotifAbstr; ASubmPaper; ANotifPap; Decision; FinalPDue
PCOrgPhases: AbstractCheck; PaperCheck; ReviewsContr; Assignment; #Reviews

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Committee</td>
<td>Member</td>
</tr>
<tr>
<td>Name</td>
<td>UserID</td>
</tr>
<tr>
<td>Type</td>
<td>Email</td>
</tr>
<tr>
<td>Fax</td>
<td>Phone</td>
</tr>
<tr>
<td>Rights</td>
<td>EmailList (virt)</td>
</tr>
<tr>
<td>MemberID</td>
<td>AbstrNotif(mat)</td>
</tr>
<tr>
<td>Email</td>
<td>PaperNotif(mat)</td>
</tr>
<tr>
<td>Address</td>
<td>...</td>
</tr>
</tbody>
</table>

**Story**

**Actor**

**Media types**

**WIS ?!?**

**Overview**

**Why**

**Foundations**

**DB Postulate**

**Services**

**Personalisation**

**Finally**
Example 2:
Submission and Reviewing System: Media Type \((PCMemberEntry)\)

parameterized expression on

\(\text{InformationMediaType}, \text{EnabledManipulationRequest}, \text{SuppliedProcesses}\)

as triple for each component of the media object

composition via

bounded iteration, sequential, branching, conditions, ...

SequentialExpression with Parameters \((\text{userID}, \text{pwrd})\)

\(\text{object} (\text{Conference head})\)

PC Member Work Sheet

- (media type \(\text{AssignedPapers}\), \(\emptyset\), \(\emptyset\))
- (media type \(\text{Missing Reviews}\), \(\emptyset\), \(\emptyset\))
- (media type \(\text{ReviseSubmitted}, \text{ReplaceByRevision}, \{\text{Select, RevisionLink}\}\))
- (media type \(\text{ViewSubmitted}, \emptyset, \emptyset\))

if \(\text{CommitteeMember} \in \text{chair}\) then (media type \(\text{ReviewByPC}, \emptyset, \text{LinkToRbPY}\))
if \(\text{CommitteeMember} \in \text{chair}\) then (media type \(\text{ReviewStatus}, \emptyset, \text{LinkToRS}\))
if \(\text{AllReviewsCompleted}\) then (media type \(\text{AnomymousReview}, \emptyset, \text{LinkToAR}\))
if \(\text{CommitteeMember} \in \text{chair}\) then (media type \(\text{AssignReviewers}, \emptyset, \text{LinkToAssRev}\))
if \(\text{CommitteeMember} \in \text{chair}\) then (media type \(\text{ViewAllPapers}, \emptyset, \text{DownloadPaper}\))
if \(\text{time} \in \text{BrowseOpen}\) then (media type \(\text{BrowseSubmittedAbstracts}, \emptyset, \emptyset\))
if \(\text{time} \in \text{BrowseOpen}\) then (media type \(\text{BrowseTitlesOnly}, \emptyset, \emptyset\))
Example 2: Paper Submission and Reviewing System: Services

Media Type (View, Functionality (, View Interfaces))

MediaType ConferenceInformation(ConfXYZ, LinkSet)
MediaType AssignedPapers(
    AssignedReview(*, CommitteeMember(MemberID)),
    DownloadPaper)
MediaType MissingReviews(if MissingReviews(memberID)
    then for(...) else Thanks, ∅)
MediaType ReviseSubmitted(SubmittedReviews = for(AssignedReview(), Review), ∅)
MediaType ViewSubmitted(SubmittedReviews = for(AssignedReview(), Review), ∅)
MediaType ReviewsByPC()
MediaType ReviewStatus()
MediaType AnonymousReview()
MediaType AdHocSQL()
MediaType AssignReviewers()
MediaType ViewAllPapers
MediaType BrowseSubmittedAbstracts(∅, getAbstractList \ forbidden)
MediaType BrowseTitlesOnly(∅, getAbstractList \ forbidden)
MediaType ConferenceInformation(ConfXYZ, LinkSet)
Example 2: Paper Submission and Reviewing System: Media Object
(submitForm)

Structure (pID,pwrd)

object(Conference head)

Submission form for papers

First part - Submission of your paper

Format selection for your paper - select one of possible choices

Location of your paper  
- browse ∨ type (mandatory)

Second part - check/correct the data you submitted with abstract

Contact person

Title of paper = f(Paper(Title))

Author_list (#, first name, last name, name of company or institute)

PC Member (is any of the authors member of the PC) (yes/no)

Remarks

Submit data for step 2

reset (either)

Functions

observe_correction(Part#2)

transfer(“address/file”)

collect(Address,Name,Format,Extension)

insert_PaperBody(pID,format,Body,today)

observe_correct(Part#2) → update_Paper(pID,Authors,Title,Email)
Example 2: Paper Submission and Reviewing System: Dialogue Scene

\[ v_1 = \sigma_{PaperID=pID}(PaperBody) \]

\[ \alpha = \text{on LinkPaperSubmit if deadline = ok } \land \text{Login = ok} \]
\[ \quad \text{accept on submitConfirmed } \land \neg \text{collectError} \]

\[ \beta = \text{on submitPaper if paperBody = ok accept on extension = ok} \]
\[ \gamma = \text{accept on } \neg \text{transferError} \]
\[ \delta = \text{on submitPaper if 1 accept on confirmed} \]
\[ A = \text{actor[pID,pwrd]} \]
Example 2: Mapping to Workflow-Specification, e.g. BPMN 1.0

Example Initial Paper Submission for Paper submission and reviewing system

Overview
Why
WIS ?!?  
Foundations
DB Postulate
Services
Personalisation
Finally

Media types

WIS

Story  Actor
Services Requested by Web Technologies

Generic Functions, Platforms, Services ... towards Enterprise 2.0

Overview
Why
WIS ?!
Foundations
DB Postulate
Services
Personalisation
Finally

Abstract
State
Services
22. 10. 2008
B. Thalheim

Media types

Story
Actor
The Co-Design Hexagonal Dimensions

Towards sophisticated web engineering

User and intention
Goal, application area
profile, information demand

Storyboard
Stories
tasks

Functionality
Navigation
search
work

Presentation
Interfaces
depending on the environment

Services
collaboration
group content
collective identity

Content
Data
objects
knowledge

Context
Technics
organisation

Overview
Why
WIS ?!
Foundations
DB Postulate
Services
Personalisation
Finally

Services allow context injection and is user-centered and story-centered
Main Challenge of Web 1.0, 2.0, 3.0, ...: Development of Services on top of DBS

- Database architecture distinguishes at least three layers:
  - a conceptual layer describing the database schema in an abstract way
  - a physical layer implementing the schema – not relevant for us here
  - an external layer made out of views

- For data-intensive services complete this architecture by adding operations:
  - On the conceptual layer operations are handled as database transactions
  - On the external layer the operations provide the means with which users can interact with a database
Media Types, Media Object Suite

Theoretical Basis

- Interaction types = \( (ct(M), q_M, Op_M) \)
  content type \( cont(M) \), defining query \( q_M \)
generic functions \( Op_M \) for changing the database

- Attached operations: (signature, selection type, body)
  selection type - supertype of \( ct(M) \)
e.g. generalization/specialization, reordering, browsing, linking, surveying, searching, join

- Media type: interaction type + unit extension
  + order extension + cohesion/adhesion + hierarchical versions

- Usage modelling: usage dimensions, scales, user profiles, user kind, context, session

- Container = \( (ct(C), layout(C), kind(C)) \)
  for shipping and representation
Database Systems and their Specification

Database system state \( DBS \)

\[(\text{input states } I, \text{output states } O, \text{DBMS states } E, \text{database states } D)\]

Modification programs:

\[
\text{if } I(req) \neq \lambda \land E(modify) = \text{enabled} \land I(req) \in \text{Update} \\
\text{then } I(req) := \lambda, O(errMsg) := \ldots, D := \ldots, E := \ldots
\]

Retrieval rules:

\[
\text{if } I(req) \neq \lambda \land E(retrieve) = \text{enabled} \land I(req) \in \text{Update} \\
\text{then } I(req) := \lambda, O(errMsg) := \ldots, O(answer) := \ldots
\]

DBMS control rules:

\[
\text{if } E(DBMS\text{stateChange}) \text{ and } E(modify) = \text{disabled} \\
\text{then } D := \ldots, E := \ldots
\]
Database Systems and their Specification

Database system state $DBS$

(input states $I$, output states $O$, DBMS states $E$, database states $D$)

DBS operation rules

$\text{MODIFYINPUT}(\text{request, DBMS\_state, DB\_state}),$
$\text{MODIFYOUTPUT}(\text{request, DBMS\_state, DB\_state}),$
$\text{MODIFYDB}(\text{request, DBMS\_state, DB\_state}),$
$\text{MODIFYCONTROL}(\text{request, DBMS\_state, DB\_state}),$
$\text{RETRIEVEOUTPUT}(\text{request, DBMS\_state, DB\_state}),$
$\text{RETRIEVEDB}(\text{request, DBMS\_state, DB\_state}),$
$\text{RETRIEVECONTROL}(\text{request, DBMS\_state, DB\_state}),$
$\text{CONTROLLERDBMS}(\text{DBMS\_state, DB\_state})$, and
$\text{CONTROLLERDB}(\text{DBMS\_state, DB\_state})$.

modify : $(\text{req}, _, s, d) \mapsto (\_\_\_, \text{errMsg}, s', d')$

retrieve : $(\text{req}, _, s, d) \mapsto (\_\_\_, \text{answ} \cup \text{errMsg}, s, d)$

controller : $(\_\_\_, s, d) \mapsto (\_\_\_, s', d')$
**Layers of Database Systems Specification**

**Level 1** (Point of view of business users): Database system defined by three states: input state, database state, output state. The *input state* is based on algebraic structure with ground terms defined on the values and the names. The *database state* is based on the (object-)relational structure with well-defined composition operators. The *output state* is a general database defined on the values and names.

**Level 2** (Conceptual point of view): Database systems are defined as an extension of level 1 by transactions, constraints, views and integrity maintenance.

**Level 3-1** (Logical point of view): The logical database system defined as an extension of level 2 by states of the database management system by the transaction and recovery engine, by the synchronization engine, the logging engine and the query translating engine.

**Level 3-2** (Physical point of view): The physical database system is defined as an extension of level 3-1 by specific functions of the DBMS.

**Level 4** (DBMS point of view): On level 4, the storage engine is modelled in detail with the buffers and the access engine.
Database Postulate

A **database system** DBS consists of

- a set $S$ of states, together with a subset $I \subseteq S$ of initial states,
- a wide-step transition relation $\tau \subseteq S \times S$, and
- a set $T$ of transactions, each of which is associated with a small-step transition relation $\tau_t \subseteq S \times S$ ($t \in T$) satisfying the postulates of a database transformation over $S$.

A **run** of a database system DBS is an infinite sequence $S_0, S_1, \ldots$ of states $S_i \in S$:

- starting with an initial state $S_0 \in I$
- for all $i \in \mathbb{N}$ $(S_i, S_{i+1}) \in \tau$ holds
- for all $i \in \mathbb{N}$ there is a transaction $t_i \in T$ with a finite run $S_i = S_i^0, \ldots, S_i^k = S_{i+1}$ such that $(S_i^j, S_i^{j+1}) \in \tau_{t_i}$ holds for all $j = 0, \ldots, k - 1$. 
Abstract State Services (ASS)
Based on the Extended View Postulate

An Abstract State Service (ASS) consists of

- a database system DBS, in which each state $S \in S$ is a finite composition $S_d \cup V_1 \cup \cdots \cup V_k$, and
- a finite set $V$ of (extended) views.

Each view $v \in V$ is associated with a database transformation such that for each state $S \in S$ there are views $v_1, \ldots, v_k \in V$ with finite runs $S_d = S_0^j, \ldots, S_{n_j}^j = S_d \cup V_j$ of $v_j$ ($j = 1, \ldots, k$).

Each view $v \in V$ is further associated with a finite set $O_v$ of (service) operations $o_1, \ldots, o_n$ such that for each $i \in \{1, \ldots, n\}$ and each $S \in S$ there is a unique state $S' \in S$ with $(S, S') \in \tau$.

Furthermore, if $S = S_d \cup V_1 \cup \cdots \cup V_k$ with $V_i$ defined by $v_i$ and $o$ is an operation associated with $v_k$, then $S' = S_d' \cup V'_1 \cup \cdots \cup V'_m$ with $m \geq k - 1$, and $V'_i$ for $1 \leq i \leq k - 1$ is still defined by $v_i$. 
Foundations in a Nutshell

Postulates for database transformations: sequential time, abstract state, background, exploration boundary, genericity

First-order structures for state description and semantics

Abstract state postulate for abstraction from all machine, coding, ... assumptions

Background of a computation based on background classes

Location operators for abstraction from storage

Update sets
Service Integration

- The integration of ASSs aims at replacing two given ASSs by a new one that supports the functionality of both original ASSs.

- Start with an integration of database systems DBS\(^1\) and DBS\(^2\):
  - Use another set \(S\downarrow\) of states together with projection functions \(p : S^1 \rightarrow S\downarrow\) and \(q : S^2 \rightarrow S\downarrow\).
  - \(p\) and \(q\) should be invariant under isomorphisms, i.e. isomorphic states are to be mapped to isomorphic states.
  - State integration requires the existence of a set \(S\uparrow\) of integrated states together with projection functions \(\bar{p} : S\uparrow \rightarrow S^1\) and \(\bar{q} : S\uparrow \rightarrow S^2\) that are “universal” in the following sense:
    - The diagram defined by \(p\), \(q\), \(\bar{p}\) and \(\bar{q}\) permutes, i.e. \(p \circ \bar{p} = q \circ \bar{q}\) holds.
    - For any other set of states \(S\) together with projections \(p' : S \rightarrow S^1\) and \(q' : S \rightarrow S^2\) that satisfy \(p \circ p' = q \circ q'\) there exists a unique function \(r : S \rightarrow S\uparrow\) with \(\bar{p} \circ r = p'\) and \(\bar{q} \circ r = q'\).
Effects on Transactions and Views

- This “pullback” definition is that it carries over to the transactions and the transition relations:
  - If $S_1$ is the start state of a transaction $t \in T^1$, then we have a transition $(S_1, S'_1) \in \tau^1$ that is defined by a run of $t$
  - Let $S \in S^{↑}$ be a state that results from integrating $S_1$ with $S_2 \in S^2$, then the corresponding function $\bar{p}$ maps $S$ onto $S_1$
  - There is a state $S' \in S^{↑}$ that results from integrating $S'_1$ with $S_2$
  - The pair $(S, S')$ is the natural extension of $(S_1, S'_1)$ to a transition on states $S^{↑}$

- The definition also allows us to preserve views:
  - Let $v$ be a view on $S^1$, which transforms a state $S^1_d$ into a state $S^1_d \cup V^1$
  - If $S_d$ is a state after integrating $S^1_d$ with some state $S^2_d$ originating from $DBS^2$, then $S_d \cup V^1$ results from integrating $S^1_d \cup V^1$ with $S^2_d \cup V^1$
Integrating Views

- Not only interested in preserving views, but in integrating them, which can be approached in the same way:
  - For views $v^1$ on DBS$^1$ transforming $S^1$ into $S^1 \cup V^1$ and $v^2$ on DBS$^2$ transforming $S^2$ into $S^2 \cup V^2$ we can first integrate $S^1$ and $S^2$ into the integrated state $S$
  - This turns $v^1$ and $v^2$ both into views over $S$
  - We can then separately integrate $V^1$ and $V^2$ into $V$, which means that we can replace $v^1$ and $v^2$ by an integrated view that will transform $S$ into $S \cup V$
- Finally, operations associated with a view carry over to the views after integration, as they merely induce a transaction and a change to the active views
Service Composition

- Composition of ASSs does not aim at replacing any existing ASS.
- The goal is to define new services that exploit functionality of existing ones.
- We have to extract components from existing ASSs and recompose these components.

Let $\mathcal{A} = (DBS, \mathcal{V}) = (S, \tau, \{\tau_t\}_{t \in T}, \{(v, \{o_1, \ldots, o_{n_v}\})\}_{v \in \mathcal{V}})$ be an ASS. A component of $\mathcal{A}$ is an ASS $(S, \tau, \{\tau_t\}_{t \in T}, \{(p_v \circ v, \{o'_1, \ldots, o'_{n_v}\})\}_{v \in \mathcal{V}'})$ with $\mathcal{V}' \subseteq \mathcal{V}$ and $\{o'_1, \ldots, o'_{n_v}\} \subseteq \{o_1, \ldots, o_{n_v}\}$.
Parallel Composition of Components

Let \( A^i = (S^i, \tau^i, \{\tau_t\}_{t \in T^i}, \{(v, \{o_1, \ldots, o_{n^i}\})\}_{v \in V^i}) \) \((i = 1, \ldots, n)\) be ASSs. Their parallel composition \( A^1 \oplus \cdots \oplus A^n \) is an ASS that is defined as follows:

- The set of states is the sum \( S = \{S_1 \cup \cdots \cup S_n \mid S_i \in S^i\} \).
- The wide-step transition relation \( \tau \) is defined by parallel composition, i.e. \((S_1 \cup \cdots \cup S_n, S'_1 \cup \cdots \cup S'_n) \in \tau \) iff \((S_i, S'_i) \in \tau^i \) for all \( i = 1, \ldots, n \).
- The set of transactions is the product \( T = \{t_1\|\ldots\|t_n \mid t_i \in T^i\} \).
- Small step transition relations are defined by parallel composition, i.e. \((S_1 \cup \cdots \cup S_n, S'_1 \cup \cdots \cup S'_n) \in \tau_{t_1\|\ldots\|t_n} \) iff \((S_i, S'_i) \in \tau_{t_i} \) for all \( i = 1, \ldots, n \).
- The set of views is also defined as a product \( V = \{v_1\|\ldots\|v_n \mid v_i \in V^i\} \).
- The sets of service operations are defined by parallel composition \( O_{v_1\|\ldots\|v_n} = \{o_1\|\ldots\|o_n \mid o_i \in O_{v_i}\} \).
Service Personalisation

- How can the selection process (in component extraction) be tailored so that out of the views and associated operations on offer only those are selected that are relevant for the intended use?
- Concentrate on the service operations treating the views they are associated with as necessary basis.
- To support the automatic or semi-automatic selection of service operations from a given ASS we have to know how it is supposed to be used.
- For this purpose we associate an action scheme or plot with an ASS.
- A plot will be an algebraic expression composed out of the service operations together with Boolean pre- and postconditions that prescribe meaningful sequences of operations.
Plots

Let $\mathcal{O}$ denote the set of service operations associated with an ASS $\mathcal{A}$, and let $\mathcal{C}$ be a set of Boolean conditions. The set $\mathcal{P}$ of plots over $\mathcal{O}$ and $\mathcal{C}$ is the smallest set with $\mathcal{O} \cup \mathcal{C} \cup \{0, 1\} \subseteq \mathcal{P}$ satisfying the following conditions:

- For $p, q \in \mathcal{P}$ we also have $pq \in \mathcal{P}$, $p + q \in \mathcal{P}$, $p \parallel q \in \mathcal{P}$ and $p^* \in \mathcal{P}$.
- For $p \in \mathcal{P}$ not involving any operation in $\mathcal{O}$ we also have $\bar{p} \in \mathcal{P}$.

Assumptions for plots:

- A Boolean condition is combined with an operation that tests it
- $\mathcal{P}$ must satisfy the axioms of Kleene algebras with tests
Preference Rules

Define preference rules by means of equations on $P$, e.g.:

- $\alpha(p + q) = \alpha p$ means that under the condition $\alpha$, if there is a choice between $p$ and $q$, then $p$ will be preferred.
- $p(q + r) = pq$ means that after $p$, if there is a choice between $q$ and $r$, then $q$ will be preferred.
- $\alpha p^* = \alpha p$ means that under the condition $\alpha$ the preference is to execute $p$ exactly once instead of iterating it arbitrarily often.
- $\overline{\alpha} p = 0$ means that $\alpha$ is a precondition for $p$.
- $p \overline{\alpha} = 0$ means that $\alpha$ is a postcondition for $p$.

Together with the conditional equations that define the axioms for Kleene algebras with tests we can use the given plot $p$ and a postcondition $\beta$ that we want to reach (it could simply be 1), and apply the equations as term rewriting rules to turn $p\beta$ into a simpler form, say $p'$. 

Concluding

- **Languages**: Define languages that satisfy the postulates / capture ASSs

- **Service Composition**: Further explore ways to compose ASS components

- **Personalisation**: Lift the rewriting approach from a propositional level to one that uses conditions on views

- **Applications**: Elaborate in detail how ASSs can be used for web warehousing, composition of WISs and other web services, etc.

- **“Applied” Applications**: Elaborate case studies for ASS use in web interoperability
From Simple Web Applications to Web Services

Web 1.0: author driven, publish/provide story/support or advertise/wait/attract/react/retain for users: inform/subscribe/obtain/answer/come back

Web 2.0: user driven, content centered, GoogleAdSense, Flickr, Wikipedia, blogs, optimised search engines, pay per click, web services, participate instead be attracted, tagging, syndication common usage of bookmarks, clicks; communities; tracking goals of usage; data ownership, portability, economics, transparency; architectures of participation Aal principle: andere arbeiten lassen
Thank you!

thalheim@is.informatik.uni-kiel.de
ASM in a Nutshell (1): States

A signature $\Sigma$ is a finite collection of function names.

- Each function name $f$ has an arity, a non-negative integer.
- Nullary function names are called constants.
- Function names can be static or dynamic.
- Every ASM signature contains the static constants $undef$, $true$, $false$.

A state $\mathcal{A}$ for the signature $\Sigma$ is a non-empty set $X$, the superuniverse of $\mathcal{A}$, together with an interpretation $f^A$ of each function name $f$ in $\Sigma$.

- If $f$ is an $n$-ary function name of $\Sigma$, then $f^A : X^n \rightarrow X$.
- If $c$ is a constant of $\Sigma$, then $c^A \in X$.
- The superuniverse $X$ of the state $\mathcal{A}$ is denoted by $|\mathcal{A}|$.

Relations are functions that have the value $true$, $false$, $undef$.

$$\forall a \in R \iff R(a) = true.$$

$$dom(f^A) = \{(a_1, \ldots, a_n) \in |\mathcal{A}| \mid f^A(a_1, \ldots, a_n) \neq undef\}$$

The superuniverse can be divided into subuniverses represented by unary relations.
ASM in a Nutshell (2): Updates

A location $\mathcal{A}$ is a pair $l = (f, (a_1, ..., a_n))$

- $f^\mathcal{A}(a_1, ..., a_n)$ - content of the location in $\mathcal{A}$
- $\{a_1, ..., a_n\}$ - set of elements of the location $\mathcal{A}(l)$

An update for $\mathcal{A}$: $u = (l, v)$

- trivial if $v = \mathcal{A}(l)$
- update set: set of updates

An update set $U$ is consistent, if it has no clashing updates, i.e., if for any location $l$ and all elements $v, w$ if $(l, v), (l, w) \in U$ then $v = w$

The result of firing a consistent update set $U$: new state $\mathcal{A} + U$

$$(\mathcal{A} + U)(l) = \begin{cases} v & \text{if } (l, v) \in U \\ \mathcal{A}(l) & \text{if there is no } v \text{ with } (l, v) \in U \end{cases}$$

for all $l$ of $\mathcal{A}$
ASM in a Nutshell (3): Models and Transitions

A state $A$ is a model of $\phi$ ($A \models \phi$) if $\llbracket \phi \rrbracket_\zeta = \text{true}$ for all variable assignments $\zeta$ for $\phi$.

A transition rules $\Sigma$

- Skip rule: $\text{skip}$
- Update rule: $f(s_1, ..., s_n) := t$
- Parallel execution rule: $P \parallel Q$
- Conditional rule: $\text{if } \phi \text{ then } P \text{ else } Q$
- Let rule: $\text{let } x = t \text{ in } P$
- For all rule: $\text{forall } x \text{ with } \phi \text{ do } P$
- Choose rule: $\text{choose } x \text{ with } \phi \text{ do } P$
- Sequence rule: $P \text{ seq } Q$
- Call rule: $r(t_1, ..., t_n)$
ASM in a Nutshell (4): Abstract State Machine

A **rule declaration** for a rule name $r$ of arity $n$ is an expression

$$ r(x_1, ..., x_n) = P $$

where

- $P$ is a transition rule and
- the free variables of $P$ are contained in the list $x_1, ..., x_n$.

An **abstract state machines** $M$ consists of a

- a signature $\Sigma$,
- a set of initial states for $\Sigma$,
- a set of rule declarations,
- a distinguished rule name of arity zero called **main rule name** of the machine.

The transition rule $P$ yields the update set $U$ in a state $A$ under the variable assignment $\zeta$: yields($P,A,\zeta,U$).

Semantics of transition rules defined in a calculus by rules:

$$ \frac{Premise_1, ..., Premise_n}{Conclusion} \text{ Condition} $$
Further Reading /1


(2) **Binemann-Zdanowicz, A., Schulz-Brücken, B., Thalheim, B., and Tschediedl, B.** Flexible e-payment based on content and profile in the e-learning system DaMiT. In *Proceedings of E-Learn 2003* (Phoenix (Arizona), USA, 2003), Association for the Advancement of Computing in Education.


Further Reading /2


(9) Hui Ma, Schewe, K.-D., and Thalheim, B. Integration and Cooperation of Media Types. ISTA 2005: 139-153


Further Reading /3


Further Reading /4


(27) Schewe K.D. and Thalheim B. Usage-Based Storyboarding for Web Information Systems Preprint 0613, Department of Computer Science, Kiel University, November 2006
Further Reading /5


(32) Schewe, K.-D., and Thalheim, B. Term Rewriting for Web Information Systems - Termination and Church-Rosser Property. WISE 2007: 261-272


Further Reading /6


(43) Thalheim, B. Component development and construction for database design. Data and Knowledge Engineering, July 2005, Vol. 54, Issue 1, 77-95