ASM Support for Validation of Specifications: Lessons Learned from an eGovernment Project

Aleksander Binemann-Zdanowicz <binemann@is.informatik.uni-kiel.de>
Zalan Kramer <kza@informatik.uni-kiel.de>
Peggy Schmidt <pesc@is.informatik.uni-kiel.de>
Bernhard Thalheim <thalheim@is.informatik.uni-kiel.de>

Kiel University
Computer Science and Applied Mathematics Institute
Olshausenstr. 40, 24098 Kiel, Germany

Keywords Abstract State Machines, behaviour anticipation, Content Management, eGovernment, software specifications, storyboarding, SiteLang

Abstract Validation of complex bulk application specifications has always been a challenge. It is usually pursued by implementing a prototype. This is particularly the case when developing complex, information-intensive web applications, e.g. eGovernment or e-learning. Valuable resources are thus invested, often just in order to realise the incorrectness of the specification. An alternative would be a specification methodology allowing for validation without causing such costs.
Websites can be specified by *storyboarding*. A storyboard describes an information system by means of its behaviour with respect to the user and to database operations. It can be refined and transformed automatically with preserved semantics into an executable ASM. We apply operational semantics of ASM for application validation. On ASM level we can also perform stepwise refinement of the specification.
The advantages of our ASM-based approach are its direct applicability and correctness. The ASM execution and validation can be performed upon a given infrastructure surrounding the application, e.g. web servers, database systems etc. The ASM serves thus as a fully functional mock-up application for the specified business process.

1 Introduction

The increasing complexity of bulk applications we experience nowadays, for instance in e-government and e-learning areas, imposes growing complexity modeling methods we have to face. In particular, the validation methodology for complex systems that are to be integrated into given existing infrastructure (e.g. a web server or a database management system) must consider the properties of the infrastructure. Thus we need not only a mechanism for transforming an abstract model to just any prototype, but also to validate this generated prototype against the given real-life infrastructure requirements.

One of our recent projects has been focused on implementing website support for eGovernment processes in cooperation with the Kiel City Council. The difficulty was given by the particular organisational heterogeneity, resulting in complex 'general rule books' defining the rules of conduct for each of the tasks pursued by local government authorities. In real life, such rule books are used in conjunction with internal knowledge about (often not fully quantifiable) competence disparities. Thus, superficial adaptability of process specifications is crucial when developing website support for eGovernment. At least one software solution must be provided that supports adaptation with respect to current situation, user, network, history, i.e. to the current context. For an eGovernment website it thus needs to be proved whether a feasible solution exists. Thus, we need a proof of concept and implementability for eGovernment websites providing the adaptability to context as mentioned above.

Our solution is to increase the efficacy of software validation. We allow the user to be involved earlier in the software development cycle, and involve the environment behaviour determined by the given infrastructure (e.g. web server, database management system) into the validation process. Our approach allows to automatically generate an executable ASM realising the abstract application specification on early stages of specification refinement. Validation is thus performed on earlier stages of software development. Thanks to the integration of the validated model into the given infrastructure, we achieve a particularly close insight into the behaviour of the specified system.

2 Storyboarding

Storyboarding techniques are used for specifying software behaviour. Storyboarding is a notion developed originally in linguistics and in movie business [5]. The story of interaction between the user and the website is the 'intrigue' or 'plot' of the narrative work or an account of events. Stories can be played in different scenarios. A story space is a set of scenarios. A story space [5] can be modeled by means of a many-dimensional graph:

Story-space(Site) $= (\ \{ \text{ scene } \}\ ,\ E\ ,\ \lambda,\ \kappa)$

transitions $E \subseteq \{ \text{ scenes } \} \times \{ \text{ scenes } \}$

$\lambda\ :\ \{ \text{ scene } \}\ \rightarrow\ \text{sceneDescription}$

$\kappa\ :\ E\ \rightarrow\ \text{events} \times \text{ifCond} \times \text{postCond}$

A scene is an abstraction distinguishing a certain set of activities. A scene is associated with a media object, with a set of involved actors, context of the scene, applicable representation styles and a dialogue step expression for the specification of the scene [5,2]. A media object is a possibly complex object generated from a database. It corresponds to a generalised view [9]. Typical media objects are images, sounds, videos, texts, simple web pages and so forth, they consist of 'atomic' objects stored in a database. A scenario is a run through a system by a set of actors. A scenario can be sequential or cyclic.

Consider the process of pursuing a business trip as in Figure 1. We have the tasks *travel authorization request*, *approval of business trip*, and *accounting and archiving*, and two actors being in employee and superior roles.

The storyboarding paradigm has been implemented as the methodology SiteLang [5,2]. It allows to specify the behaviour of a web information system by means of user and database behaviour in parallel. SiteLang specifications can be stepwisely refined and made executable on different specification stages.

3 The SiteLang Engine

3.1 Goals

The aim of our approach is to enable validation of web-based information systems. Thus, on the basis of a SiteLang specification we generate an ASM engine with the following properties. It has to implement the desired functionality as expressed on the business user layer and on the conceptual layer [8]. Moreover, it has to be universal enough to be integrated easily into varying infrastructures. Typically changing parts of the infrastructure are web servers (e.g. Apache, Tomcat) and database management systems (e.g. Oracle, Sybase, MySQL).

The engine must be able to communicate with the user and with the existing infrastructure (web server, database system) by accepting input streams of events and by generating database transactions as well as user dialogues. The way the engine communicates with the database management system must enable the realisation of transactional behaviour of user interaction by applying the transaction mechanism of the database [8].

The dialogue generation performed by the engine must comprise typical dialogue constructs known from website development. The dialogue elements must be described by means on conceptual level, so that the presentation and layout specification can be exchanged without having to modify the engine.

Finally, the ASM engine must allow to support multiple actors performing varying utilisation scenarios at a time. Hence, the engine must be able to control multiple database transactions and maintain the synchronisation between scenarios running in parallel.

3.2 Functions of the Engine

The ASM engine is generated automatically upon a SiteLang specification we have designed. The generated ASM code is made executable by the Xasm compiler [1]. In order to satisfy the requirements we have mentioned in the above subsection, the ASM realises the following functionality.

1. *On-the-fly generation of dialogues.*
   The dialogue definition usually varies while the user interacts with the system. For instance, in order to achieve adaptable content playout, dialogue steps performed by the user can trigger modifications of the dialogue [10].
2. *Extraction of content from the database.*
   Content is given by media objects, i.e. views defined on the database. They are used to fill the dynamically generated dialogues with data.
3. *Modifying content.*
   The view definition may vary over time. When a user performs a certain action, this may result in the necessity of redefining his/her needs and filling the modified views with data anew.
4. *Initialising the database.*
   Initial database schema as well as database content must be specified. The engine generates corresponding database requests in order to reset the database to the desired starting point.
5. *Executing scenes and dialogue steps.*
   State changes of the user interaction machine and the workflow machine

must be maintained correspondingly. Scenes are maintained consistently along with the transient and persistent data. Dialogue steps are fired by the user according to the event-condition-action principle [5]. The engine thus provides mechanisms ensuring the proper execution of user interaction.

6. *Realisation of the transactional behaviour on user interaction side.* Transactions on user interaction provide a measure for rolling back user operations leading to unwanted results [3]. The ASM engine allows to map user transactions to the ones of the database management system.

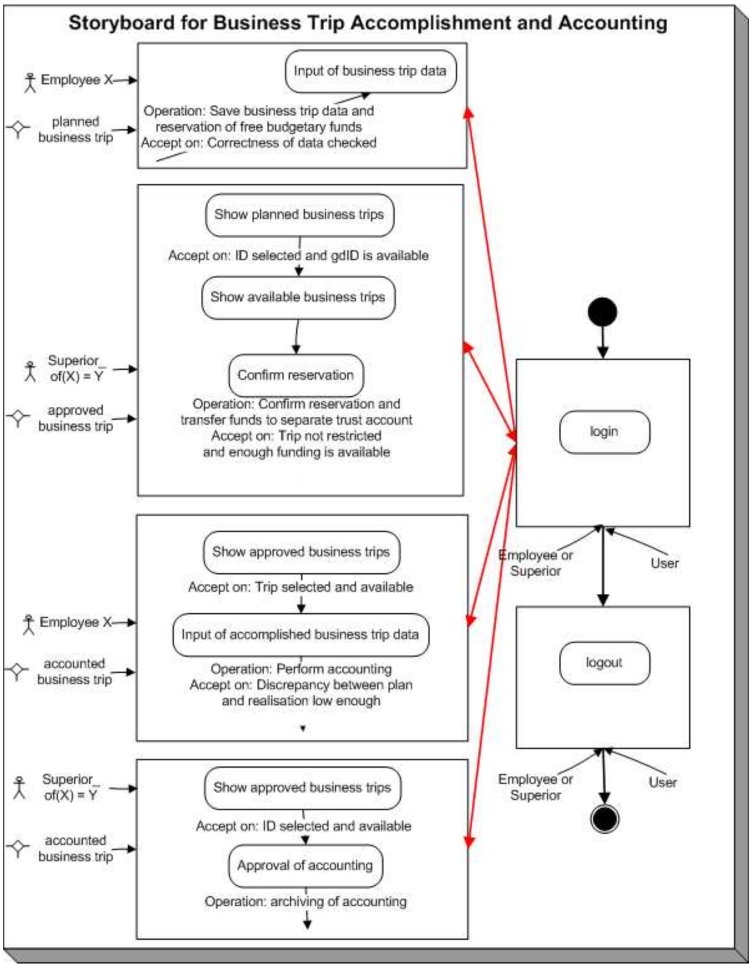

Figure 1. Storyboard of a business trip application

3.3 Integration into Existing Infrastructure for Validation

The ASM engine communicates with the environment by means of output functions and monitored functions [1]. In our example we model a web-based, content-intensive information system. Thus, the engine communicates with the a web server and a database management system. We validated the ASM engine with the web server *Tomcat* and the DBMS *Oracle9*.

The communication is organised as presented in Figure 2. The ASM engine generates dialogues for the users. The dialogues are delivered to the web server and displayed to users. The users react in form of events, which are then sent back to the ASM engine. In parallel the ASM engine communicates with the DBMS. Queries for filling media objects (views) with data, as well as update queries for data manipulation are sent to the database. The DBMS returns result sets the ASM engine can use for dialogue generation. It also realises the transactional behaviour of the ASM engine, i.e. user interaction transactions [7] are mapped by the ASM engine onto DBMS transactions. The communication with Tomcat and the one with the DBMS allow for asynchronous events coming from both of the infrastructure components.

The actual infrastructure components the ASM engine communicates with can be changed easily, as Figure 3 shows. It is done just by changing the definitions of *monitored* and *output* functions realising the communication. These functions are written in C. The rest of the ASM engine remains unchanged. An example usage of an output function for generating database queries looks as follows (the queries are sent to the DBMS in XML form).

```
external "C" [output] function write_to_buffer_DBMS ⟶ String
write_to_buffer_DBMS := "<insert><into>PERSON</into><attr>NAME</attr>
                          <value>Schmidt</value>...</insert>"
```

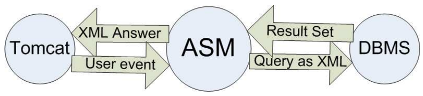

Figure 2. Integration of the ASM engine into given environment

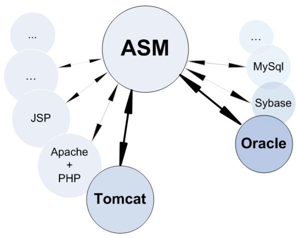

Figure 3. Integration of the ASM engine into varying infrastructures

4 Solution through ASM

Our modeling approach for information systems comprises the conceptual development of workflow and interactivity in parallel. The conceptual model is then mapped to the workflow and interaction model in SiteLang. This model is then compiled by the SiteLang compiler to an ASM specification. The latter one is a parallel composition of the following machines:

1. initialisation machine (executed *once*) ASM^{Init}:
   - database (schema and data) initialisation,
   - defining scene and initial media objects,
   - initial definitions of dialogues;

2. transaction control machine ASM^{TA}:
   - state machine for DBMS control and data flow,
   - generation of database queries and transformation from QBE to XML,
   - processing of returned DBMS transaction results;
3. scene execution machine ASM^{ScEx}:
   - scene destroying and creation,
   - content object querying control,
   - dialogue contruction mechanism;
4. dialogue step processing machine ASM^{DStep}:
   - dialogue step finishing machine,
   - machines for event processing for each dialogue step in each scene;
5. input event stream machine ASM^{inEv}:
   - delivering DBMS events (result sets and transaction stati) to ASM^{TA},
   - delivering user events (i.e. user reactions) to ASM^{DStep}.

The machines listed above are composed parallelly. The synchronisation of the machines is realised by ASM function updates, and by modifications of the database content. The latter has turned out to be necessary in order to benefit from the transactional behaviour of a proper database management system.

In the following we can see an example rule of the ASM machine realising the dialogue step processing mechanism [5,7].

choose event in $EventsUniverse$ **with** $StateOf(event) = fresh\_event$
  if <precondition>(event) **is true**
  extend $TransactionsUniverse$ **with** t
    $Add\_Dialogue\_Step\_Operations\_to\_Transaction(t)$
    $Define\_Postcondition\_for\_Transaction(t)$
    $TransactionInvoked(t) := true$
    $StateOf(event) := waiting\_for\_DBMS$

5 Implementation

5.1 The Sitelang Framework

In order to compile a Sitelang specification into ASM, a cross-compiler has been developed [2,3]. It generates an executable ASM engine, which communicates with the given DBMS and the given web server. The communication is realised by means of POSIX threads, socket and signal communication in C under Linux and Solaris, as well as Java Native Interface. Each of the external components of the given infrastructure (web server, database) has a wrapper realising the communication. The wrapper for the DBMS is realised by means of JDBC and is in the final testing phase.

In order to help the user develop a well-defined Sitelang specification, a graphical editor is being developed [6].

5.2 Generation of Executable Specifications

As mentioned above, our starting point is a conceptual specification of interactivity and application workflow. Having obtained a SiteLang specification of these both aspects, it can be compiled to an ASM specification. The latter one can be compiled to an executable one. Thanks to the operational semantics we are able to validate the behaviour of our system before its actual implementation.

A big advantage of our approach is the possibility to validate specifications not only by means of a running program, but also to integrate it with little effort into an existing infrastructure. Therefore the executable specification can be used as a mock-up application for information system we are designing. Once we have realised during the validation that our model does not satisfy the requirements, we can change the specification easily and observe the changed behaviour of the mock-up application straight away.

We can also refine the SiteLang specification as in Figure 4 [4]. The refinement may affect the interaction flow as well as the database behavior, i.e. views defining the content.

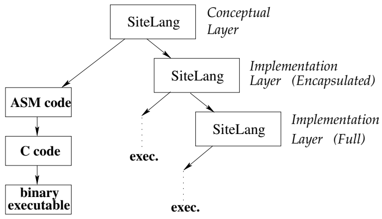

Figure 4. Refining a SiteLang specification

5.3 Dialogue Generation

The ASM engine offers a variety of constructs for building scene dialogues. It comprises static text fields, complex fields with text generated from the database, pictures generated from the database, buttons, list boxes and so on. Dialogue constructs can be made 'clickable' and return events on user's action. The events are customisable and are processed according to the SiteLang specification.

5.4 Proof-of-Concept Application

We have used the business trip application motivated by our eGovernment project as in Figure 1 for testing our approach. The storyboard in Figure 1 has been described as a SiteLang specification and compiled to an ASM engine. In Figure 5 we can see a screenshot of the running ASM engine. The generated user dialogue is delivered to the user by the Tomcat web server and presented in a web browser.

6 Future Work

At the moment the ASM engine supports a pre-defined set of dialogue constructs on conceptual level. To support any type of dialogue constructs, a dialogue construct schema should be developed. The ASM machine would thus be extended to implement the behaviour of user-designed dialogue constructs.

Dialogue states are represented in the ASM engine by means of transient function values. Our project experience has taught us it would be advantageous to have the ASM engine make every dialogue state change persistent.

7 Conclusion

We propose a novel approach to validating complex web-based applications. It is based on automatic generation of an ASM engine serving as a mock-up application. The conceptual basis for engine generation is the storyboard paradigm for website modeling. The ASM engine is a fully functional implementation of of the modeled application, including the entire user interaction flow and the database behaviour. It is able to communicate with given infrastructure components, e.g. web servers and database systems. The operational semantics of ASM allows for immediate validation of the application at the respective refinement level. Our mock-up approach has been successfully tried out in an eGovernment project currently realised in cooperation with the Kiel City Council.

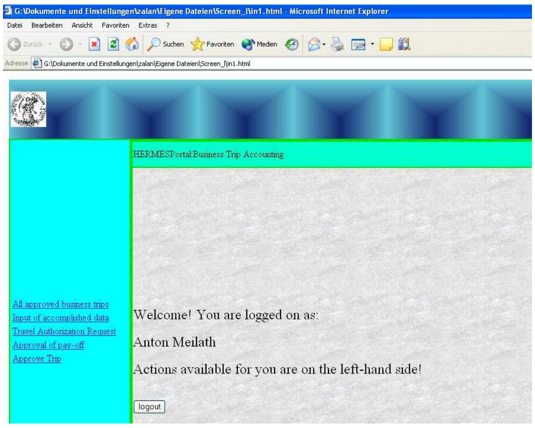

Figure 5. Example dialogue generated in the business trip application

References

1. M. Anlauff. An extensible component-based asm language (abstract state machines - theory and applications, lncs 1912). In *Abstract State Machines - Theory and Applications, LNCS 1912*, pages 69–90, Berlin, 2000. Springer.
2. A. Binemann-Zdanowicz. Towards information system modeling on the basis of asm semantics. In *Computer Science Report I-12/2001, Brandenburg University of Technology at Cottbus*, 2001.
3. A. Binemann-Zdanowicz and B. Thalheim. Modeling information services on the basis of asm semantics. In *Proc. ASM'2003, Lecture Notes in Computer Science 2589*, pages 408–410. Springer, 2003.
4. E. Börger and R. Stärk. *Abstract state machines - A method for high-level system design and analysis*. Springer, Berlin, 2003.
5. A. Düsterhöft and B. Thalheim. Conceptual modeling of internet sites. In *S. Kunii et al. (Eds.): Proc. of ER 2001, LNCS 2224*, pages 179–192, Berlin, 2001. Springer.
6. G. Fiedler, T. Raak, I. Romalis, K.-D. Schewe, and B. Thalheim. Website modeling, website orchestration, and website management. In *LIT'04*, pages 219–228. infix Verlag, 2004.
7. S. Srinivasa. *An Algebra of Fixpoints for Characterizing Interactive Behavior of Information Systems (PhD)*. BTU Cottbus, 2001.
8. B. Thalheim. Informationssystem-entwicklung. In *BTU Cottbus, Computer Science Institute, Technical Report I-15-2003*, Cottbus, 2003.
9. B. Thalheim and K.-D. Schewe. Web information systems: Usage, content, and functionally modelling. In *Technical report 0405, Christian Albrecht University at Kiel, Institute of Computer Science and Applied Mathematics*, April 2004.
10. B. Tschiedel, A. Binemann-Zdanowicz, and B. Thalheim. Logistics for learning objects. In *Proc. eTrain'2003, Pori, Finland, July 2003.* Kluwer, 2003.