Association for Information Systems
AIS Electronic Library (AISeL)

Wirtschaftsinformatik Proceedings 2005 Wirtschaftsinformatik

February 2005

A Concept for Modelling and Validation of Web Based Presentation Templates

Daniel Fötsch
Friedrich-Schiller-Universität Jena

Andreas Speck
Friedrich-Schiller-Universität Jena

Wilhelm R. Rossak
Friedrich-Schiller-Universität Jena

Jörg Krumbiegel
Intershop Communications AG

Follow this and additional works at: http://aisel.aisnet.org/wi2005

Recommended Citation
Fötsch, Daniel; Speck, Andreas; Rossak, Wilhelm R.; and Krumbiegel, Jörg, "A Concept for Modelling and Validation of Web Based Presentation Templates" (2005). *Wirtschaftsinformatik Proceedings 2005*. 21.
http://aisel.aisnet.org/wi2005/21

This material is brought to you by the Wirtschaftsinformatik at AIS Electronic Library (AISeL). It has been accepted for inclusion in Wirtschaftsinformatik Proceedings 2005 by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

In: Ferstl, Otto K, u.a. (Hg) 2005. *Wirtschaftsinformatik 2005: eEconomy, eGovernment, eSociety*; 7. Internationale Tagung Wirtschaftsinformatik 2005. Heidelberg: Physica-Verlag

ISBN: 3-7908-1574-8

© Physica-Verlag Heidelberg 2005

A Concept for Modelling and Validation of Web Based Presentation Templates

Daniel Fötsch, Andreas Speck, Wilhelm R. Rossak
Friedrich-Schiller-Universität Jena

Jörg Krumbiegel
Intershop Communications AG

Abstract: The assurance of quality and reliability is essential for success in the e-business. However, missing validation mechanisms are a serious problem in web page development. Most web based programming languages do not support validation and other security features. Furthermore, the usage of different languages increases the complexity. We present an approach to integrating different programming languages in one homogeneous language and further in a formal model, which may be transferred to the specific model used by verification tools. Our concept enables the single analysis and validation of heterogeneous web based languages as well as in combination and interaction.

Keywords: Modelling, Validation, Model Checker, Template

1 Introduction

In the early days of the Internet web designers used HTML to create static web pages composed of text and images. However, static web pages must be manually created and maintained. Naturally, this is practical only for comparative small web sites containing just a few pages of information, which only need to be changed from time to time.

In contrast to a simple, often non-commercial web site, an online shop possibly contains thousands of different pages making it impractical to create and maintain all the storefront pages manually. In the Intershop's e-commerce system Enfinity MultiSite a specific concept has been introduced to handle the large numbers of web pages: the template concept. Without such a template concept, storefronts require that every page be hard-coded in standard HTML.

In general, **templates** contain information about how to format or present data. This skeleton may be developed in the form of a rule or mould with one or more shapes used to apply different looks and feels to specific documents and web sites.

Furthermore, templates may include instructions to replace placeholder variables with current values or content from another source. The technique of templates supports the following significant benefits:

- A template can be used to display variations, e.g. thousands of products.
- Calculations or other processing flows are not possible in standard HTML.
- Information, which are subject to regular changes, such as product prices, may be stored and maintained in a database.
- A storefront page may display user-defined pieces of information, for example a list of products matching a customer's search criterion.

Comprising with templates, it is possible to generate storefront pages dynamically each time a user requests a page. This dynamic procedure increases the complexity considerably. Ensuring the high stability and quality of a web presence, an automatic verification concept and technique is essential.

Admittedly, templates have some oppressive characteristics for an automatic verification. The fundamental problems to resolve are:

- Generally, templates are developed with heterogeneous web based programming languages, e.g. HTML, JavaScript, and other different server-side scripting languages.
- Web based scripting languages are supported by interpreters, but they cannot be used for an overall analysis.
- The examination of linked web pages is very difficult and currently not enough supported.
- Especially, no concept exists for the verification of the mutual interaction from pipelines and templates in Enfinity MultiSite.

We develop a concept of a multi-level transformation process from templates implemented with different programming languages in a formal model. The formal model is the basis for the further conversion in a specific model of verification tools [Föts04].

In this paper we concentrate on the technique of model checking. Model checking is an automatic approach to formal verification. This verification is performed by software tools, which are capable of deciding whether or not a formal specification is satisfied by a given model. In this context the model is a state-transition system and the specification is formalised with temporal logic formulas, which pinpoint desired behaviour over paths and states in the model [Cla+01].

In the following of the paper we apply our developed transformation process to the template concept of Intershop's e-commerce system Enfinity MultiSite. Hence, we introduce in section 2 the programming layers of Enfinity MultiSite and demonstrate a use case model of an HTTP request. Based on this background

information, the detailed steps of the transformation process are discussed in section 3. At the beginning of this section our formal model to be checked is defined. In the following subsections we explain the physical description of the defined model and depict an example from the e-commerce domain. In section 4 we elaborate the problem of validating. We pick up again our example for describing the conversion from the defined model into a specific model of a model checker. Finally, we consider related works, conclude this work, and provide an insight into our future work.

2 Enfinity MultiSite

Enfinity MultiSite is a large e-commerce system implemented by Intershop. Enfinity MultiSite distinguishes four different programming layers (cf. Figure 1). Each programming layer has a distinct function, contains distinct objects, and is programmed in dedicated development tools [Schw03].

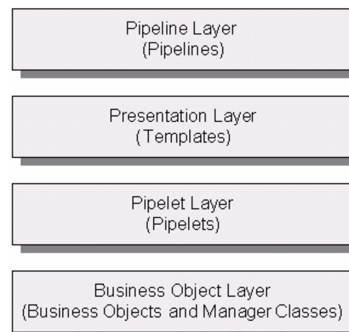

Figure 1: The programming layers of Enfinity MultiSite [Schw03, p. 3].

1. *Pipeline Layer*: The highest level of the layered Enfinity MultiSite model realises the concept of a pipeline. A pipeline is a graphical model of a particular business process and combines a number of business tasks using a determined syntax to delineate the business process. Every request to Enfinity MultiSite triggers the execution of a pipeline. The requested URL always identifies, which pipeline is to be invoked to serve the request. A pipeline may include different types of nodes: pipelet node (implements the reusable specific business function), control node (defines the flow of a pipeline) and interaction node (generates a response). Pipelines are created, managed, and manipulated within the Visual Pipeline Manager.

2. *Presentation Layer*: The presentation layer is responsible for converting the results of a business process into a response. Typically, this results in a generated HTML page being displayed in the end user web browser. In this context the templates define the skeleton of this web page: the page structure, the style used on the page, and static as well as dynamic content elements. The dynamic content is based on information stored in the pipeline dictionary and added to the page at the time the response is actually generated. The inclusion of variable elements and expressions for dynamic content in templates allows to derive a large number of pages from each template.

3. *Pipelet Layer*: Based on functionality offered by the business object layer, pipelets handle a specific business function, such as connecting to the database and retrieving information. Pipelets are small, reusable units of Java classes, which can be used in different pipelines. Pipelets exchange data with other pipelets via the pipeline dictionary of the pipeline.

4. *Business Object Layer*: The primary building blocks of the business object layer are the business objects and manager classes. Typically, business objects are data containers modelling data persistently stored in the database. Each business object is connected to a manager class. Manager classes provide services for controlling the lifecycle of persistent data object contained in the business objects. Business objects and managers are implemented with EJB technology.

The concept of Enfinity MultiSite's programming layers has two significant benefits:

- The business logic is managed outside the code. The pipelets implementing the logical business function are combined to perform a pipeline representing the logical business flow.

- The business logic is separated from the design of presentation. Although pipelines and templates work together, the tasks of modelling business logic can be performed separately from developing the design of the storefront.

Figure 2 depicts an overview of how an HTTP request is processed in Enfinity MultiSite and illustrates the interaction of the key concepts mentioned above.

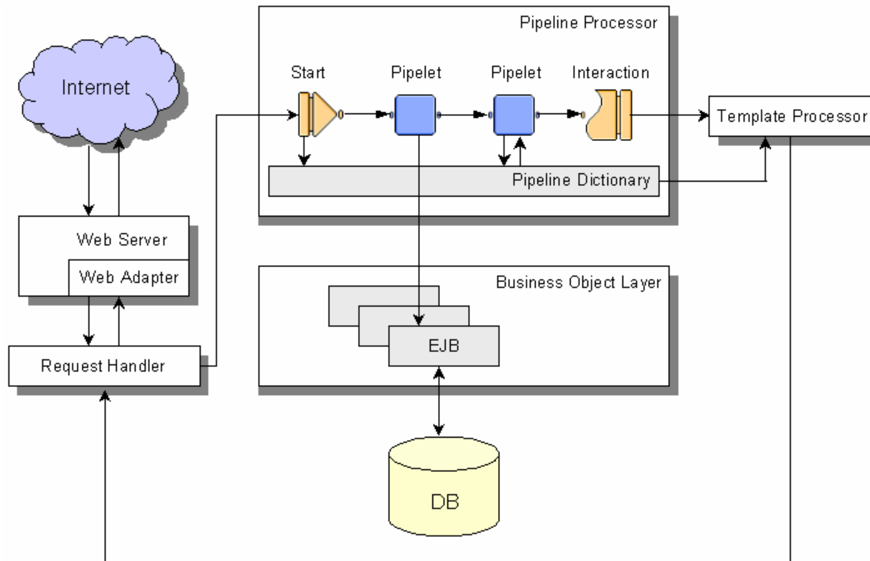

Figure 2: Simplified Enfinity MultiSite request model (according to [Mül+02, p. 69])

At first, the web server receives the HTTP request of the client. The web server augmenting with a web adapter adds supplemental information, such as session id, to the incoming request and redirects the enhanced request to the core application server of Enfinity MultiSite.

The application server consists of a request handler, a pipeline and a template processor, as well as further components. The request handler is responsible for the encapsulation of incoming and outgoing requests in appropriate request, response, and session objects. After some data are examined, e.g. specific currency and language supported by the online shop, the request handler invokes the pipeline processor, which tries to load the pipeline encoded in the requested URL. Moreover, the pipeline processor initiates the pipeline dictionary and manages the flow of the pipeline elements through the pipeline. At the end of a pipeline the pipeline processor invokes an interaction node for producing a response. The interaction node with the referenced template triggers the template processor. Using the data stored in the pipeline dictionary, the template processor generates an HTML page, which displays the results of the pipeline [Mül+02].

3 Modelling of Presentation Layer

The basis of the process of transformation is a common generic language. Each heterogeneous template implementing with different languages is transformed in a

homogeneous template. We use XML for representing the homogeneous template. XML is a standardised extensible markup language [Bra+], which is a subset of SGML (standard generalized markup language [Inte86]). XML allows easy specification of user-defined markup tags adapted to the document [Ahm+01].

In the next step of the transformation process the homogeneous XML template is transformed in a formal model. This model (called presentation model) provides a basis for converting in the specific model of model checker or other verification tools.

3.1 Definition of the Presentation Model

We will now define the formal definition of the **presentation model**. The formal model for the presentation layer is an 8-tupel $M = (S, E, C, L, S_0, S_E, \delta, \lambda)$, where

- S is a finite set of states,
- E is a finite set of events,
- C is a finite set of conditions,
- L is a finite set of labels,
- $S_0 \subseteq S$ is a distinguished set of initial states,
- $S_E \subseteq S$ is a distinguished set of final states,
- δ is a transition function from $S \times E \times C$ into S,
- λ is a label function from S into L.

The states of the finite automaton represent computations and statements in the templates, which are performed in dependency of the transitions in a specific sequence. The execution of the transitions can be influenced themselves with conditions and events. Events, e.g. press a link, are responsible for triggering a transition. Conditions specify if the transition is performed or not. Each automaton representing a template begins with an initial state and ends with a final state. In addition, states can be occupied with a label, which contains more information about the state (for example the name of a variable element).

The presentation model will be described as graphical notation in the further paper (cf. Figure 3). The UML state charts diagram notation is extended to an optional label in the states, a grey area for optional states, and macro states. Thereby, a macro state is the shortened notation for various non-macro states. The labels are marked as little rectangle in the corresponding state and the grey area in the background of the states signs that either all or no states can be omitted.

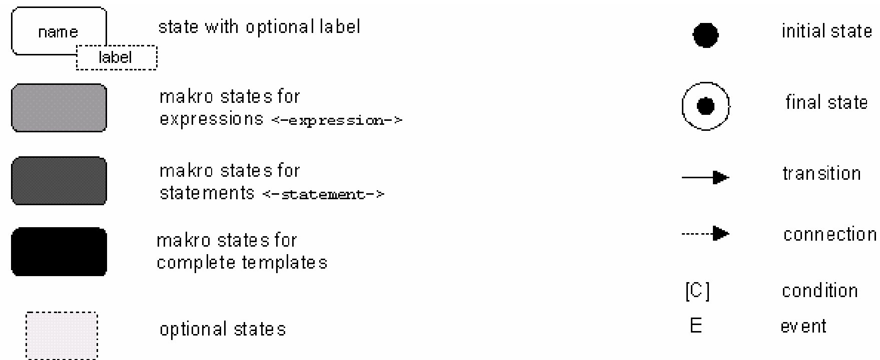

Figure 3: The graphical notation of the presentation model.

3.2 Physical Description of the Presentation Model

The physical description of the presentation model is realised in XML. XML supports several helpful features for validating a presentation model. For instance, an XML document is valid if it is well formed, the document must simply conform to numerous syntactical rules, and adheres to its specified document type definition (DTD). A **document type definition** is a formal description of a grammar to define the legal building blocks of an XML document. A DTD can be declared inline in the XML document, or as an external reference. On this note, a presentation model of a template is valid if and only if it fulfils following DTD:

```
<!ELEMENT presentation-model (states, transitions)>
<!ELEMENT states (state+)>
<!ELEMENT state EMPTY>
<!ATTLIST state
    kind CDATA #REQUIRED
    label CDATA #IMPLIED
    id ID #REQUIRED
    idref IDREF #IMPLIED>
<!ELEMENT transitions (transition*)>
<!ELEMENT transition (from, to, event?, condition?)>
<!ATTLIST transition>
```

```
<!ELEMENT from EMPTY>
<!ATTLIST from
    idref IDREF #REQUIRED>
<!ELEMENT to EMPTY>
<!ATTLIST to
    idref IDREF #REQUIRED>
<!ELEMENT event EMPTY>
<!ATTLIST event
    kind CDATA #REQUIRED>
<!ELEMENT condition EMPTY>
<!ATTLIST condition
    value (true|false) #REQUIRED>
```

According to above DTD, there are nine element types. The main `presentation-model` element must contain exactly one `states` followed by exactly one `transitions` element. The `states` element represents the states in the presentation model. Therefore, the `states` element is a non-empty set of `state` elements. Each `state` is specified with a specific `kind` attribute (e.g. comment), an `id` attribute, an optional `label` attribute (e.g. do something), and if necessary an `idref` attribute. The `id` is required to assign the states to the correct transitions. In distinctive cases (e.g. a variable assignment) states are in relationship with other states. This can be specified with the optional `idref` attribute.

The `transitions` element maps the set of transitions to the presentation model. Thus, the `transition` must contain one `from` and one `to` element. The `from` element comprises a reference to the starting state and the `to` element to the ending state of the transition. In addition, a `transition` element can be possessed an `event` as well as a `condition` element. These elements represent the events and conditions in the transitions of the presentation model. Note that the order of the elements must be as specified.

3.3 Example of Modelling

In this section we demonstrate an example transformation from the original template in the graphical notation of the presentation model.

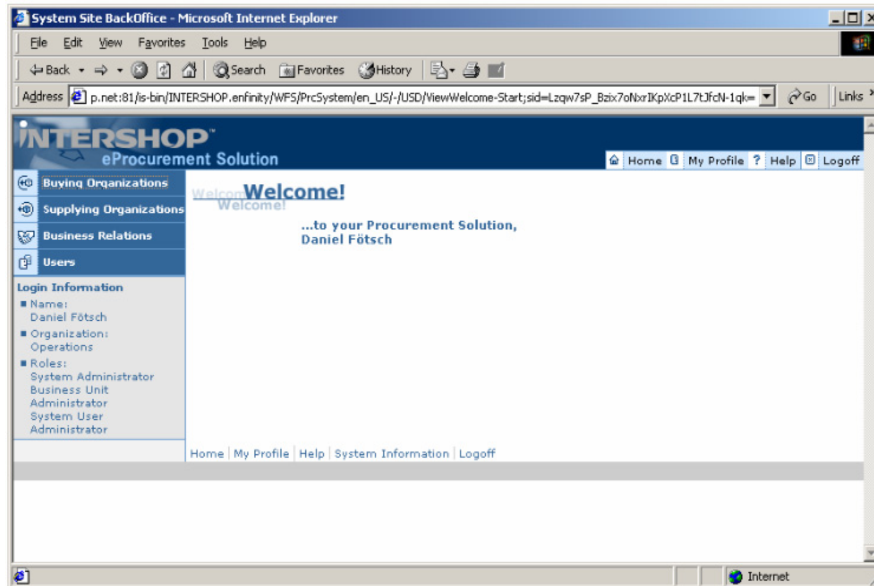

Figure 4: The Welcome web page from Enfinity's eProcurement solution.

Our example web page (cf. Figure 4) is generated from the template `ApplicationFrame`. All templates in Enfinity are stored in ISML (Intershop markup language). ISML is an extended markup language based on Java Server Pages technology, which is primarily used to add dynamic content to a page. ISML allows to retrieve and display data from the pipeline dictionary, to initiate the execution of additional pipelines, and to embed templates inside other templates [Inte02].

The example `ApplicationFrame` template, which generates the web page in Figure 4, consists of four templates, the `GlobalNavigationBar`, the `SiteNavigationBar`, the `Footer` and the `#Working Template#`. Moreover, the `ApplicationFrame` template includes a `GlobalJava Script` file and a stylesheet file.

In the first step of the transformation all heterogeneous files of the `Application Frame` template are transferred in a common generic XML file. In the next step this XML file is transformed in an automaton of the presentation model. Figure 5 shows a fragment of this XML file in graphical notation. The file of the presentation model is the basis for the following transformation in the specific model of model checker or other verification tools.

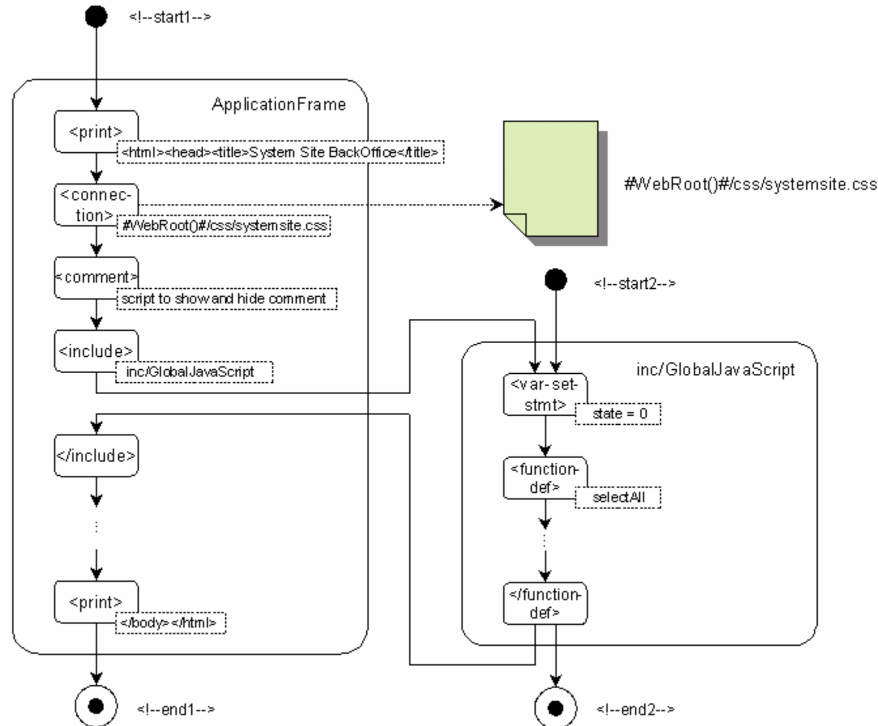

Figure 5: A fragment of the graphical presentation model of the example web page.

The example in Figure 5 outlines the most important characteristics of the transformation. On one side each for the validation interested file (e.g. `ApplicationFrame` and `JavaGlobalScript`) is separately integrated in the presentation model. For that reason, each file can be single validated. On the other side, according to the relation of the files, they will be connected. For example the `ApplicationFrame` includes the `GlobalJavaScript` file in the presentation model. Therefore, the complete template can be validated in interaction and combination. Note, in the case that no `GlobalJavaScript` file exists in the presentation model, a transition is generated between `<include>` and `</include>` state inside the automaton of the `ApplicationFrame`.

In this way the whole presentation layer can be transformed in the presentation model and, finally, validated with a verification tool. In the next section one validation technique is applied on the presentation model.

4 Validation of Presentation Layer

In the further process of transformation the presentation model has to be converted in the specific model of a verification tool.

4.1 Validation with Model Checker

We apply model checkers, such as SPIN [Holz97] or SMV (Symbolic Model Validation [McMi92]), in order to validate the templates. These checking tools need a model of the issue to be validated (the possible behaviour of the system) usually described as finite state machine. The specification (the desired behaviour of the system), which the model is checked against, is formulised in temporal logic formulas. If an error is recognized the model checker provides a counter-example consisting of a scenario in which the model behaves in an undesired way. Figure 6 depicts the verification methodology of model checking.

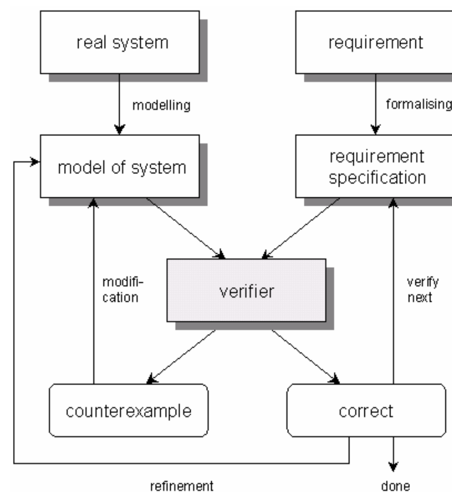

Figure 6: The concept of model checking [Kato99, p. 32].

For the objective of specification several temporal logics with different power exist, e.g. the LTL (Linear Time Logic [Pnue77]) or the CTL (Computation Tree Logic [ClDr89]). These logics enhance common predicate logics by defining additional operators. For LTL the temporal operators X (next time), F (eventually), G (globally) specify that a property holds at the second/some/each state on a path. Additionally, CTL defines the path quantifiers A (for all computation paths) and E (some computation paths). Furthermore, the binary operator U (until) can be de-

fined for both LTL and CTL. A comparison between linear and branching time logic is introduced in [EmHa86] or [ClDr89].

4.2 Example of Validation

The example presented in Figure 5 of section 3.3 is a comparative small fragment of the complete template as recently as the whole presentation layer. However, it demonstrates some of the problems, which may occur when a template has to be examined.

The transformation from the presentation model to the finite state machine of a model checker, such as SMV, is straightforward. Each state and each transition of the presentation model is wrapped in a state as well as a transition in the SMV automaton. The conditions and the events influencing the transition can be inherited from the automaton as additional guards. Merely, the information of labels in the presentation model is currently not supported by the model of the model checker. They get lost. But these information are not crucial for the dynamic behaviour of the template.

The description of the requirement specification to be checked against the model of SMV is formalised in CTL. Examples for concrete rules to be checked may look like:

- On a page (represented by the state `start1`) may be displayed a specific text (label of the state `print1`). In CTL: `AG(start1 -> EF(print1))`
- In the case the user has a specific permission (condition of `if1` is true) a specific text (`print2`) must be displayed. In CTL: `AG(if1 & (if1_condition=true) -> AF(print2)).`
- Validate, if a linked template (represented by the URL address in the label of the state `include1`) is included in the presentation model. In CTL: `AG(include1 -> !EX(endinclude1))`

The procedure of template validation is depicted in Figure 7. Here the template is transformed into the corresponded finite state machine. The specification is expressed in CTL (the last specification example above). Both are given to the model checker (actually they are copied into one file, which is then processed by SMV).

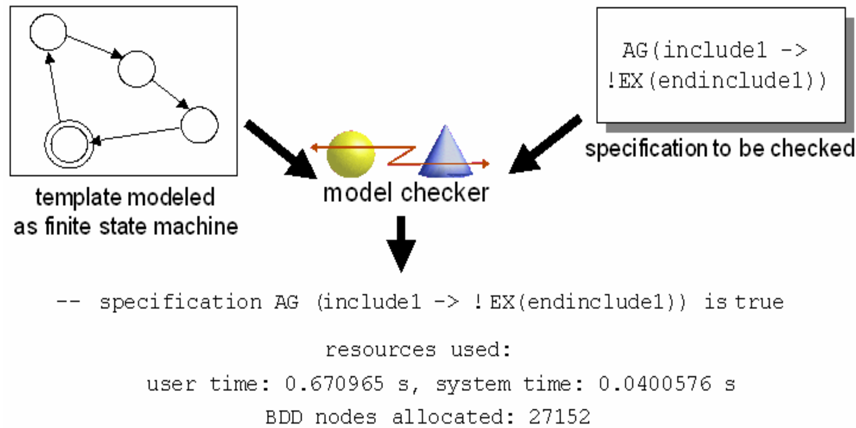

Figure 7: Model checking of templates.

The result produced by the model checker is presented at the bottom of Figure 7; the positive validation that no next state `endinclude1` of the state `include1` exists. In other words, this means that the included template is integrated in the presentation model of the example.

5 Related Work

The concept of applying XML to store information is very convenient in e-commerce systems (e.g. like Intershop's Enfinity MultiSite). It may also be used for representing programming languages.

JavaML realises such an approach representing the elements of the programming language Java in XML [Badr00]. Thereby, the diversity of XML tools may be used for analysing and processing of Java code.

In our work we combined both the representation of static web sites and the representation of a programming language in one concept: applying XML to store the dynamic elements of web presentations (modelled as automatons) in XML.

In order to validate the XML automatons we apply model checking. This is an approach, which is well established in hardware-related domains. Model checking is widespread and has already industrial relevance since several years [ClKu96]. The usage of this formal method in the domain of software products, however, is still in its very beginnings (the first steps may be found in [LaGr98]). This is due to the state explosion problem as well as the model construction problem. Both are even more difficult to deal with software as compared to hardware systems [Cor+00].

The transformation from state charts to the model checking language SMV is systematically investigated in [ClHe]. Despite the fact that state charts are usually quite close to the code, this approach deals with similar problems (building finite automaton from dynamic models).

The validation of the behaviour of components is also related to this work since it meets the problems of large presentations. In [StWa01] an approach called PACC is presented. It allows component certification. The approach considers to enforce predefined and designed interaction patterns and is therefore based on comparable software analysis and documentation techniques. The focus in their work is on certification and documentation. Another approach to model and validate the dynamic activities of components may be found in [Spe+022]. In this approach model checking is explicitly applied in order to validate the behaviour of the components and composites.

6 Conclusion and Future Work

This paper proposes an approach to model and validate web based presentation templates. The modelling and validation is realised by applying a multi-level transformation process. The basis of the process is the transformation of the heterogeneous languages in a common generic XML language. Then the generic language is mapped in a formal presentation model. This model provides the fundament for the conversion in the specific model of model checkers or other verification tools.

Our approach has been used to develop web presentation templates for the e-commerce system Enfinity MultiSite and to validate their correctness. However, the concept presented in this paper may be also applied for other web systems, which use dynamic mechanisms, such as templates, for generating web pages. We expect it to also support the validation of dynamic web pages created by e.g. PHP or other scripting languages without any major adjustments.

From the viewpoint of the Enfinity MultiSite's programming layers the paper presents only a limited solution for the validation of the presentation layer. However, further validation techniques have to be found to include the pipeline and pipelet layer for assuring their consistency. Our current steps towards this issue are the investigations of formal modelling pipelines and pipelets, the integration of this model in the presentation model, as well as the development of an e-commerce specific specification language allowing to express the standard requests in the e-commerce domain. The latter work implies that the temporal specification languages have to be considered and a mapping from these languages to the standard classes of problems have to be built.

References

[Ahm+01] Ahmed, K.; Ancha, S.; Cioroianu, A.; Cousins, J.; Crosbie, J.; Davies, J.; Gabhart, K.; Gould, S.; Laddad, R.; Li, S.; Macmillan, B.; Rivers-Moore, D.; Skubal, J.; Watson, K.; Williams, S.; Hart, J.: Professional Java XML. Wrox Press Ltd.: Birmingham, 2001.

[Badr00] Badros, G. J.: JavaML: A Markup Language for Java Source Code. http://www.cs.washington.edu/research/constraints/web/badros-javaml-www9.pdf, 2000, Download 2004-10-01.

[Bra+04] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E.; Yergeau, F.: Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation. http://www.w3.org/TR/REC-xml, 2004, Download 2004-10-01.

[ClDr89] Clarke, E. M.; Draghicescu, I. A.: Expressibility results for linear-time and branching-time logics. In Proceedings of the Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency, School/Workshop. Lecture Notes in Computer Science, Springer: London, 1989, pp. 428-437.

[Cla+01] Clarke, E. M.; Grumberg, O.; Peled, D. A.: Model Checking. The MIT Press: Cambrige, Massachusetts, 2001.

[ClHe00] Clarke, E. M.; Heinle, W.: Modular translation of statecharts to SMV. Technical Report CMU-CS-00-XXX, School of Computer Science, Carnegie Mellon University, Pittsburgh, 2000.

[ClKu96] Clarke, E. M.; Kurshan, R. P.: Computer-aided Verification. IEEE Spectrum 33(6), 1996, pp. 151-178.

[Cor+00] Corbett, J. C.; Dwyer, M. B.; Hatcliff, J.; Laubach, S.; Pasareanu, C. S.; Robby; Zheng, H.: Bandera: Extracting Finite-state Models from Java Source Code. In Proceedings of the 22nd International Conference on Software Engineering (ICSE'00), 2000, pp. 439-448.

[EmHa86] Emerson, E. A.; Halpern, J. Y.: "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. Journal of the ACM 33(1), 1986, pp. 151-178.

[Föts04] Fötsch, D.: Modellierung und Validierung von web-basierten Sprachen. Diploma thesis, Computer Science Department, Friedrich Schiller University of Jena and Intershop Research, Jena, 2004.

[Holz97] Holzman, G. J.: The Model Checker SPIN. IEEE Transaction on Software Engineering 23(5), 1997, pp. 279-295.

[Inte86] International Organization for Standardization (ISO). Standard Generalized Markup Language (SGML). ISO 8879:1986. http://www.iso.ch/cate/d16387.html, 1986, Download 2004-10-01.

[Inte02] Intershop Communications AG: Enfinity Content Management. Technical White Paper. http://www.intershop.de/pdf/products/ECM_WhitePaperT_020523.pdf, 2002, Download 2004-10-01.

[Kato99] Katoen, J.-P.: Concepts, Algorithms, and Tools for Model Checking. Lecture Notes of the Course "Mechanised Validation of Parallel Systems", Course number 10359. Department of Computer Science, Friedrich Alexander University of Erlangen-Nürnberg, 1999.

[LaGr98] Laster, K.; Grumberg, O.: Modular Model Checking of Software. In Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98). Lecture Notes of Computer Science, Springer: London, 1998, pp. 20-35.

[McMi92] McMillan, K. L.: Symbolic Model Checking: An approach to state explosion problem. PhD thesis CMU-CS-92-131, School of Computer Science, Carnegie Mellon University, 1992.

[Mül+02] Müller, I.; Braun, P.; Rossak, W. R.: Integrating Mobile Agent Technology into an e-Marketplace Solution, the InterMarket Solution. Technical Report No. 2002-06, Computer Science Department, Friedrich Schiller University of Jena, 2002.

[Pnue77] Pnueli, A.: The temporal Logic of Programs. In Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science (FOCS'77), Providence, Rhode Island. IEEE Computer Society Press: 1977, pp. 46-57.

[Schw03] Schwedler, N.: Prüfung dynamischer Ablaufketten. Diploma thesis, Computer Science Department, Friedrich Schiller University of Jena and Intershop Research, Jena, 2003.

[Spe+02] Speck, A.; Pulvermüller, E.; Jerger, M.; Francyk, B.: Component Composition Validation. International Journal of Applied Mathmatics and Computer Science 12(4). University of Zielona Gora Press: 2002, pp. 581-589.

[StWa01] Stafford, J.; Wallnau, K.: Predicting Feature Interactions in Component-Based Systems. In Proceedings of the Workshop on Feature Interaction in Composed Systems, in Association with the 15th European Conference on Object-Oriented Programming (ECOOP'01). Technical Report No. 2001-14, University of Karlsruhe, 2001, pp. 35-41.