Modelling and Verifying of e-Commerce Systems

Andreas Speck

Friedrich-Schiller-University Jena
Department of Economics
Integrated Application Systems Group
andreas.speck@uni-jena.de
www.wiwi.uni-jena.de/wi2/

Abstract. Static function hierarchies and models of the dynamic behaviour are typically used in e-commerce systems. Issues to be verifies are the completeness and correctness of the static function hierarchies, business rules valid in defined business domains and the consistency of models on different levels of abstraction. Today the systems are mostly tested manually. Automated support may the verification of static dependencies modelled in Boolean logic. Moreover, the checking of dynamic process models can be supported by temporal logic specifications and model checking tools.

1 Introduction and Problem Statement

Most large scale commercial systems share similar modelling and architecture concepts. e-Commerce systems like Intershop Enfinity are typical examples of this type of systems. Moreover other commercial e.g. the ERP system SAP R/3 have the same modelling characteristics based on business processes. The design is modelled on different abstraction levels. The static dependencies between the functionality are modelled in hierarchies. The dynamic interactions are designed in process and workflow models.

A typical modelling method for commercial systems in general is ARIS (Architecture integrated Information Systems) supported by the ARIS tool set of IDS Scheer (a closer description of a ARIS subsystem optimised for e-commerce is given in section 2). The typical problems that arise from this modelling concept are:

- Static function hierarchies must be complete and correct.
- Specific business rules, which are valid in defined business domains, are to be kept.
- Dynamic models on different levels of abstraction must be consistent.

Current practice is to test and check systems manually. Besides specific rules and regulations, experiences are used as base. However, it is obvious that the manual tests do not cover all possible errors. Additionally they are very cost intensive especially for commercial software developers.

2 e-Commerce Modelling by ARIS

As already mentioned the ARIS modelling concept is used in commercial systems in general. In this section we will present ARIS for Enfinity [1] as a typical modelling concept for e-commerce systems. Since there exist numerous modelling variants in the general ARIS concept, we will focus on this specific version ARIS for Enfinity.

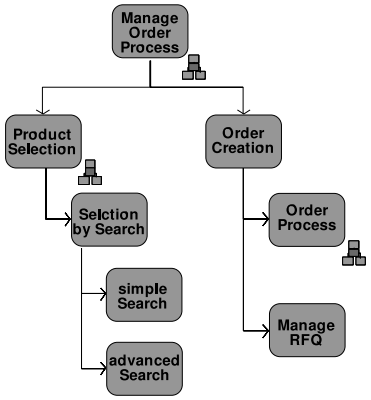
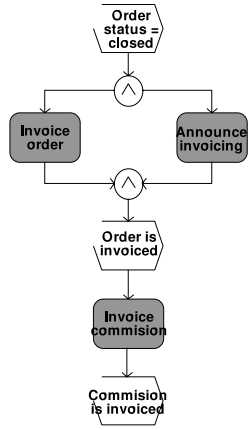
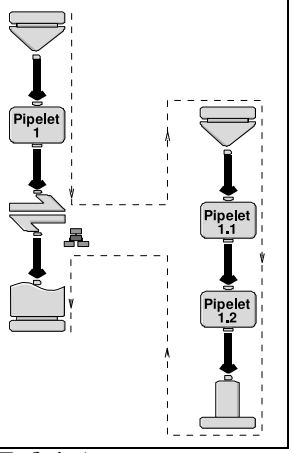
Function Hierarchy (generic model)	*Event-driven Process Chain* (generic model)	*Intershop Pipeline* (proprietary model ARIS for Enfinity)
		

Table 1. Model Types of the ARIS Method (ARIS for Enfinity)

However the ARIS for Enfinity modelling concept must still be considered as a very generic model for e-commerce systems and may be applied for other systems e.g. like mySAP. As there exist no final standards in e-commerce modelling (currently on base of Web Services and BPEL4WS first approaches towards standardisation are in progress) ARIS for Enfinity already gives a clear outlook on the future standard. Since there are clear parallels between future BPEL4WS modelling and the already successfully applied ARIS for Enfinity, this paper is based on the ARIS for Enfinity experiences.

ARIS for Enfinity supports four layers of abstraction. The layers one to three are standard in large scaled-commerce systems and other commercial applications. Only the fourth layer uses an Enfinity specific model presentation (*Intershop Pipelines*). However the basic concept is also base of generic modelling like in BPEL4WS, which bears considerable similarities to the *Intershop Pipelines*.

1. *Business Scenarios* define the basic core, coarse grain elements of the e-commerce application. They are very abstract. The verification on this level is and will be done by human experts. Automation is not desirable.
2. *Business Process Overview* provides the overview over the functionality (main functions). Usually only the static relations of the functions are considered (function hierarchies).
3. *Business Processes* describe the dynamic processes in the system. Yet, the models are abstract. However, the business process models may be refined to the concrete workflows. Usually EPCs (*Event-driven Process Chains*, [6]) are applied as modelling technique.
4. *Workflows* describe the executable processes of the system. Intershop invented a proprietary model: *Intershop Pipelines*, which are on the one hand graphical models

and on the other hand are interpreted by the application server of the e-commerce system. Modelling languages on a similar level are web-service descriptions like BPEL4WS (which are issue of standardisation at the moment).

The different ARIS model types are depicted in table 1. The *Function Hierarchy* shows which function is member of which other function.

The dynamic model *Event-driven Process Chain* is a rather abstract process description. The main elements are the functions (rounded rectangles) and the events (hexagons) which are connected by the organisational flow (arrows) and the logic connectors (Boolean `AND` as depicted as well as `OR` and `XOR`). Examples of further elements are organisational units executing certain functions, deliveries or application systems providing functions.

The *Intershop Pipeline Models* include different types of nodes: *Pipelet nodes* implement the re-usable specific business function. This business function is realised by a small Java class. For more complex operations these classes call library code (in the *persistence layer*), e.g. access to data bases or other systems. *Control nodes* define the control flow of *Pipelines*. An example in the table is the call of a *Sub-Pipeline* or the termination of this *Sub-Pipeline*. The interaction of the *Pipelines* and the web pages (*templates*) exported to the web clients is described in *interaction nodes*. These nodes define which data a *Pipeline* receives from a template activating the *Pipeline* (start interaction) and which data is delivered to the resulting *template* (interaction).

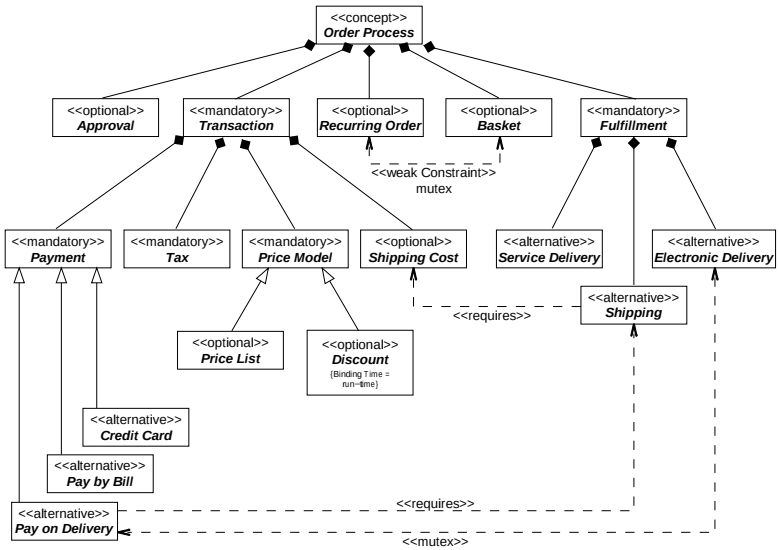


Fig. 1. UML Function Hierarchy

3 Verification Concepts

3.1 Static Function Hierarchies

The static function hierarchies are rather easy to verify. The configurations of functions may be represented by Boolean formulae [8] which may be verified automatically (e.g. by theorem proofer or term rewriting).

The more interesting question is how to visualise the hierarchy. One possibility may be to extend the ARIS model. Another alternative is to apply a more complex model. UML class models may serve as compromise. They support numerous expressive modelling elements and can be exported from the ARIS tools.

Besides the ordinary inclusion relationship (represented by aggregation) there exist variability (inheritance). Functions may be mandatory (`AND`), optional (`OR`) or alternative (`XOR`). Moreover cross-tree constraints are introduced (either both functions required or are mutual exclusive).

Typical rules on the level of static function hierarchies are:

- The number of times specific functions have to occur. E.g. a web-shop does not support all types of payment or delivery which are potentially possible.
- The locations in the hierarchy, where functions or sub-hierarchies of functions are placed, are relevant.

Depending on the specific application type (called solution) like business to business (B2B), business to consumer (B2C) or e-procurement there are variants (considered as rules) in the characteristic hierarchies, e.g. B2C and e-procurement solutions offer a comparatively large, predefined sub-hierarchy of price management functions while B2B solutions support only a single function.

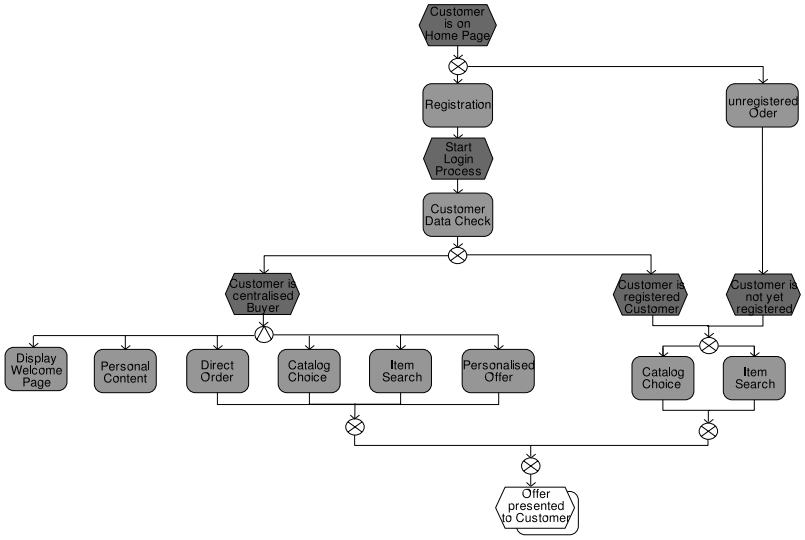


Fig. 2. Example: *Login and Presentation Process*

3.2 Business Rules in Dynamic Models

At Intershop a set of business rules (best practices) have been collected. Usually these rules are base of the manual testing. Figure 2 shows a business process to be verified. This process describes the login and identification of a customer and the resulting different purchase and presentation functions. In contrast to the most other process models the chosen example is quite small and easy to overlook. Usually the models are much larger.

These rules, which have to be verified, deal with the temporal dependencies. Therefore they can be transformed in temporal logic. In our case we apply *Computational Tree Logic* (CTL) which is supported by the model checking tool *Symbolic Model Validation* (SMV) [7]. The process models are transformed to automata verified by SMV.

1. A customer must be presented a catalogue choice in all cases. This rule is valid for all customer categories. An appropriate CTL formula representing this rule is:
   `AG(Customer_is_on_Home_Page -> AF(Catalog_Choice))`
   It is generally assumed **A**lways **G**lobally (`AG`) that the customer is already an the home page of the shop ($Customer\_is\_on\_Home\_Page = \top$). Now it has to be verified that this precondition implies **A**lways in the **F**uture (`AF`) the customer will have a *Catalog_Choice*. There is no path that will not lead to *Catalog_Choice*.
2. In this example a path has to be identified on which customer data are checked (is true). Here we have to check that there is no state where customer data are checked in order to receive a counter-example form the model checker.
   `AG(Customer_is_on_Home_Page -> AG(!Customer_Data_Check))`
   Again, we start with the same assumption as in the first example ($Customer\_is\_on\_Home\_Page = \top$).

The results of the SMV model checker are shown in figure 3.

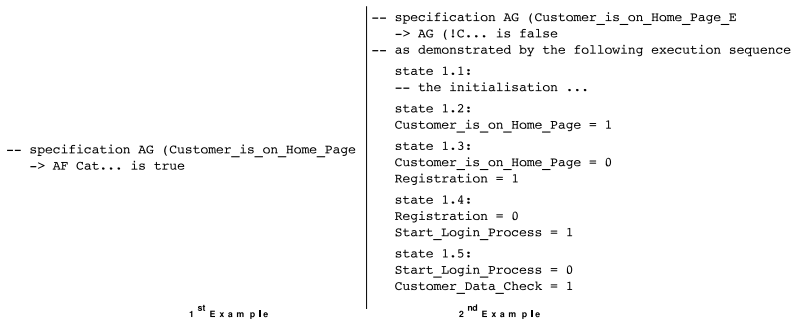


Fig. 3. SMV Checking Results

Further examples for best practices / business rules are:

- Before a customer pays always a product presentation page is shown.
- There is no path which leads to an unintended loss of the content in the customer's shopping cart.
- The customer's order may be identified in the different subsystems (ERP, logistic system or e-commerce system).

3.3 Consistency between Models on Different Levels of Abstraction

There are no typical rules for the refinement of more abstract models (e.g. EPC models) to more detailed ones (e.g. *Pipelines* or BPEL4WS). However, usually characteristic sequences in abstract models should appear in detailed models. These characteristic sequences may serve as specifications, which have to be verified. These sequences may be considered as a rules which are then checked as demonstrated in section 3.2. However up to now there is little research how to manage this kind of very specific rules which are different for each application. The versioning approach for static features like presented in [8] may be starting point for future work.

4 Related Approaches

In hardware verification there are a considerable number of model checking approaches. The verification of software and specifically the dynamic processes is still at it's very beginning. An early example for software checking is *Bandera*, an integrated collection of program analysis and transformation components [4], which allows to validate Java code to a certain extend (e.g. limited lines of code).

Assuming the system is developed starting with manually produced state charts (as realised in [3] and [2], for instance) it is possible to translate programs and requirements applying a generation technique. However, consistency problems arise if the model is produced independently from the implementation.

The validation of the behaviour of components is also related to this work since it meets the problems in the lower abstraction levels. In [10] an approach called PACC is presented. It allows component certification and documentation. The approach considers enforcing predefined and designed interaction patterns. Another approach to model and validate the dynamic activities of components may be found in [9]. In this approach model checking is explicitly applied in order to validate the behaviour of the components and composites.

Some approaches deal with the validation of workflows in general by applying model checkers. An example for a workflow checking approach is [5]. Here web service workflows are to be validated.

5 Conclusion and Future Work

The testing of large scale commercial systems such as e-commerce systems is currently manually realised. Function hierarchies may be modelled in Boolean logic which are automatically verified by theorem proofer or with term rewriting. The checking of dynamic process models (e.g. against business rules) can be supported by temporal logic and model checking techniques.

Future work will be to improve the workflow models on the most detailed level. Further semantic information has to be captured. Code characteristics of the *Pipelets* will be included in the models.

Another issue is the improvement of the model checking technology. Since the SMV is quite optimal for the representation of hardware, the support of workflows and software requires new verification systems to be invented. Specifically properties (e.g. representing further semantics) and their hierarchical arrangement have to be dealt with.

References

1. M. Breitling. Business Consulting, Service Packages & Benefits. Technical report, Intershop Customer Services, Jena, 2002.
2. E.M. Clarke and W. Heinle. Modular translation of statecharts to smv. Technical Report CMU-CS-00-XXX, Department of Computer Science, CMU, Pittsburgh, 1998.
3. K. Fisler, S. Krishnamurthi, D. Batory, and J. Liu. A Model Checking Framework for Layered Command and Control Software. In *Proceedings of the Workshop on Engineering Automation for Software Intensive System Integration*, June 2001.
4. J. Hatcliff, C. Pasareanu, R. S. Laubach, and H. Zheng. Bandera: Extracting Finite-state Models from Java Source Code. In *Proceedings of the 22nd International Conference on Software Engineering*, June 2000.
5. C. Karamanolis, Giannakopoulou D., J. Magee, and S. Wheater. Model Checking of Workflow Schemas. In *In Proc. of the 4th International Enterprise Distributed Object Computing Conference (EDOC'00)*. IEEE Computer Society, 2000.
6. G. Keller, M. Nüttgens and A.-W. Scheer. Semantische Prozessmodellierung. Technical report Nr. 89, Veröffentlichungen des Instituts für Wirtschaftsinformatik, Saarbrücken, 1992.
7. K. McMillan. Symbolic Model Checking. PhD Thesis, CMU, Pittsburgh, 1992.
8. E. Pulvermüller, A. Speck, and J. O. Coplien. A Version Model for Aspect Dependencies. In *Proceedings of 2nd International Symposium of Generative and Component-based Software Engineering (GCSE 2001)*, LNCS, Erfurt, Germany, September 2001. Springer.
9. A. Speck, E. Pulvermüller, M. Jerger, and B. Franczyk. Component Composition Validation. *International Journal of Applied Mathematics and Computer Science*, 12(4):581–589, 2003.
10. J. Stafford and K. Wallnau. Predicting Assembly from Certifiable Components. In *Proceedings of the Workshop on Feature Interaction in Composed Systems, ECOOP 2001, Technical Report No. 2001-14, ISSN 1432-7864*. Universitaet Karlsruhe, June/September 2001.