A Formal Methodology to Specify E-commerce Systems

Conference Paper · October 2002

DOI: 10.1007/3-540-36103-0_20 · Source: DBLP

CITATIONS
9

READS
249

5 authors, including:

Adriano C. M. Pereira
Federal University of Minas Gerais
161 PUBLICATIONS **1,983** CITATIONS
SEE PROFILE

Mark A. J. Song
Pontifícia Universidade Católica de Minas Gerais
66 PUBLICATIONS **166** CITATIONS
SEE PROFILE

Wagner Meira Jr.
Federal University of Minas Gerais
584 PUBLICATIONS **10,249** CITATIONS
SEE PROFILE

Sergio Campos
Federal University of Minas Gerais
123 PUBLICATIONS **2,911** CITATIONS
SEE PROFILE

A Formal Methodology to Specify E-commerce Systems

Adriano Pereira[1] Mark Song[1,2] Gustavo Gorgulho[1]
Wagner Meira Jr.[1] Sérgio Campos[1]
{adrianoc, song, gorgulho, meira, scampos}@dcc.ufmg.br
{mark}@una.br

[1]Department of Computer Science
Universidade Federal de Minas Gerais
Caixa Postal 702 - CEP 30.123-970
Belo Horizonte – Minas Gerais – Brazil
Phone Number: 55-31-34995860 Fax: 55-31-34995858

[2]UNA - União de Negócio e Administração

Abstract. Electronic commerce is an important application that has evolved significantly recently. It gives companies the possibility of reaching an unprecedented number of clients at very low cost. However, electronic commerce systems are complex and difficult to be correctly designed. Currently, most approaches are *ad-hoc*, and frequently lead to expensive, unreliable systems that may take a long time to implement. In this work we propose a methodology that uses formal-method techniques, specifically *symbolic model checking*, to design electronic commerce applications and to automatically verify that these designs satisfy properties such as atomicity, isolation, and consistency. Using the proposed methodology, the designer is able to identify errors early in the design process and correct them before they propagate to later stages. Thus, it is possible to generate more reliable applications, developed faster and at low costs. In order to demonstrate the applicability and feasibility of the technique, we have modeled and verified a virtual store in which multiple buyers compete for product items. The model verified has more than 10^{23} states and verification has been completed in few minutes. For instance, the verification process pointed out a concurrency control error which allowed the same item to be sold twice.

Keywords: electronic commerce, design specification, model checking, formal verification, property patterns

1 Introduction

E-commerce has become a popular application. In general, we can define electronic commerce as the use of the network resources and information technology to ease the execution of central processes performed by an organization.

As new e-commerce services are created, new types of errors appear, some unacceptable. We define error as any unexpected behavior that occurs in a computer program. A typical error that may occur in a site is to allow two users to buy the same item.

However, guaranteeing the correctness of an e-commerce system is not an easy task due to the great amount of scenarios where errors occur, many of them very subtle. Such task is quite hard and laborious if only tests and simulations, common techniques of system validation, are used.

Formal methods consist basically of the use of mathematical techniques to help in the documentation, specification, design, analysis, and certification of computational systems. The use of formal methods, in special model checking, is sufficiently interesting and promising once it consists of a robust and efficient technique to verify the correctness of several system properties, mainly regard to identification of faults in advance.

This paper presents a new methodology to design e-commerce systems applying model checking. The Section 2 defines some important concepts about model checking. In the Section 3, our proposed methodology is explained, and Section 4 shows an example of its use. Section 5 analyzes the related works, and Section 6 presents some conclusions and future work.

2 Model Checking of E-commerce Systems

Most electronic commerce systems can be modeled using a few entities: the products being commercialized such as books or DVDs, the agents that act upon these products such as consumer or seller, and the actions that modify the state of the product such as reserving or selling an item.

Similarly to traditional commercial systems, the main entity of electronic commerce is the product that is transactioned. For each product being commercialized there are one or more items, which are instances of the product. Each item is characterized by its life cycle, which can be represented by a state-transition graph, i.e., the states assumed by the item while being commercialized and the valid transitions between states. Examples of states are *Reserved* or *Sold*. The item's domain is the set of all states the item can be.

The entities that interact with the e-commerce system are called agents. Examples of agents are buyers, sellers and the store manager. The agents perform actions that may change the state of an item, that is, actions correspond to transitions in the life cycle graph. Put an item in the basket or cancel an item's reserve are examples of actions.

Services are sequences of actions on products. While each action is associated with an item and usually comprises simple operations such as allocating an item for future purchase, services handle each product as a whole, performing full transactions. Purchasing a book is an example of a service, which consists of paying for the book, dispatching it, and updating the inventory.

2.1 Business Rules

An e-commerce system can be described by its business rules. A business rule is a norm, denoted property, which specifies the operation of an e-commerce application.

In [4, 5] was developed a system of property specification patterns for finite-state verification tools based in the scope, order and occurrence of an event[1]. Each pattern has a scope, which is the extent of the program execution over which the pattern must hold. There are five basic kinds of scopes, as explained: Global: the entire program execution; Before: the execution up to a given event; After: the execution after a given event; Between: any part of the execution from one given event to another given event; After-until: like between but the designated part of the execution continues even if the second event does not occur.

The scope is defined by specifying a starting and an ending event for the pattern. For state-delimited scopes, the interval in which the property is evaluated is closed at the left and open at the right end. Thus, the scope consists of all states beginning with the starting state and up to but not including the ending state. A list of some patterns, with short descriptions, follows: Absence: a given event does not occur within a scope; Existence: a given event must occur within a scope; Bounded Existence: a given state must occur k times within a scope; Universality: a given event occurs throughout a scope; Precedence: an event P must always be preceded by an event Q within a scope; Response: a event P must always be followed by an event Q within a scope.

In the next subsections we present a brief background on model checking and CTL-formulas.

2.2 Symbolic Model Checking

Ensuring the correctness of the design at its earliest stage is a major challenge in any system development process. Current methods use techniques such as *simulation* and *testing* for design validation. Although effective in the early stages of debugging, their effectiveness drops quickly as the design becomes clear. A serious problem with these techniques is that they explore *some* of the possible behaviors of the system. We may never be sure whether the unexplored execution paths may contain fatal bugs. A very attractive alternative to simulation and testing is the use of *formal verification* approach, which explores exhaustively all possible behaviors of the system.

Symbolic model checking [2] is a formal verification technique by which a desired behavioral property of a system can be verified over a model through exhaustive enumeration of all the states reachable by the application and the behaviors that traverse through them. The system being verified is represented as a *state-transition graph* (the model) and the *properties* (the behaviors) are described as formulas in temporal logic. Labels are associated to the values of the variables in the program, while transitions correspond to steps in the model.

[1] In an e-commerce system, an event describes a set of actions.

In the next section we explain our methodology to incrementally design verifiable e-commerce systems.

3 The Formal Specification Methodology

There are many types of e-commerce applications, such as digital library, virtual bookstore, auction sites and others. The difference between them are their nature and their business rules. Some business rules are common, for example: an item should not be sold to more than one customer. On the other hand, there are many other rules specific to the application, as to allow or not the reservation of an item, to provide supply control, or to define priority to transactions executed concurrently.

The proposed methodology, an extension of the *CAFE* methodology [9], consist of a way to design e-commerce systems to apply model checking. *CAFE* methodology explains how to specify an e-commerce system and we consider the user should know some formal language, such as SMV [7], to build the model.

Our methodology is incremental and divided into four major levels. It is relevant to emphasize that these organization were adopted in order to simplify the design specification, but the designer may employ another organization.

The first level, defined as conceptual, embodies the business rules and the definition of the e-commerce system to be designed. As many details the designer specifies, as easier would be to apply the methodology and achieve good results in the verification process.

The second level, called application, models the life cycle of the item that is commercialized, identifying the types of operations (called actions, as we refer to henceforth) that are performed on it and change its state.

The third one, named functional, models the services provided by the system and the concept of multiple items are introduced.

The last level contemplates the components of the system and the user's interaction with them. It completes the scope of the system, modeling its architecture, so we called it the architectural level.

3.1 Properties of an e-commerce system

Properties may be described, for examples, as formulas in CTL [2]. CTL-formulas are built from atomic propositions, boolean connectives and temporal operators.

For example, a rule can describe that an item can only be reserved if it is available. To specify this property, a developer would have to translate this informal requirement into the following CTL-formula: AG (((item_state = Available) & (service = Reserve) & (product_inventory > 0) & (next(product_inventory) = product_inventory - 1)) $\rightarrow$ AX ((item_state = Reserved))).

As we can see the specification process demands some expertise in formal methods. We contend that acquiring this level of expertise represents a substantial obstacle to the adoption of the methodology. So, to overcome this problem, we suggest to use a *specification pattern system.*

Thinking about an e-commerce application model, we realize that the first important property to verify is completeness. This property guarantees that the model is consistent, by asserting that all states and actions are achieved.

To express this property we can use the property pattern of *Existence*. Additionally it is necessary to define the scope as *after*, considering that "Exists in the Future" means "after current state/event".

Transitivity is a property which defines the next state to be achieved after the occurrence of an event in the current state. It is necessary to check its veracity to guarantee the correct execution of the services that satisfy the business rules.

Most properties to be verified in e-commerce systems relate to transactions. A transaction is an abstraction of an atomic and reliable sequence of operations executed. Transaction processing is important for almost all modern computing environments that support concurrent processing. In an electronic commerce server, a transaction consists of a sequence of actions affecting the existing items, each action potentially modifying the state of the item. One of the most important properties that must be satisfied in this context is guaranteeing that the transactions being executed are consistent, that is, showing that the concurrency control mechanism implemented is correct and that concurrent transactions do not interfere with each other. In other words, we must check that transactions are atomic.

We have verified three types of properties that relate to the consistency of transactions: *Atomicity* - a transaction must be finished or not started, that is, if it does not finish, its effects have to be undone; *Consistency* - the state of the product must remain coherent at all times; *Isolation* - the result of one transaction must not affect the result of another concurrent transaction.

In the next subsections we detail the levels of the formal methodology, using real examples of e-commerce business rules to explain how this properties can be checked.

3.2 Conceptual Level

Formally, we characterize an e-commerce system by a tuple $< P, I, D, Ag, Ac, S >$, where P is the set of products, I is the set of items, D is the set of product domains, Ag is the set of agents, Ac is the set of actions and S is the set of services.

Products are sets of items, that is, $i \in I$ means that $i \in p, p \in P$. The products partition the set of items, that is, every item belongs specifically to a single product. Formally, $I = \bigcup_{\forall p \in P} p$ and $p_i \cap p_j = \emptyset$ for $i \neq j$. Domains are associated with items, that is, each item i is characterized by a domain D_i. Two items of the same product have the same domain, i.e., for all items $i, j \in I$, there is a product p such that if $i \in p$ and $j \in p$, then $D_i = D_j$.

Each action is associated with a transition in the state-transition graph of the item and is defined by a tuple $< a, i, tr > \in Ac$, where $a \in Ag$ is the agent that performs the action, and $i \in I$ is the item over which the action is performed, and $tr \in D_i X D_i$ is the transition associated with the action. In our model, the actions performed on a given item are totally ordered, that is, for each pair of actions x and y, where i_x and i_y are the same, either x has happened before

y or y has happened before x. Services are defined by tuples $< p, A >$, where $p \in P$ and $A = a_1, a_2, \ldots$ is a sequence of actions such that if $a_i = (d_1, d_2)$, $a_{i+1} = (d_3, d_4)$ then $d_2 = d_3$ $\forall i, d_i \in D_j$ where D_j is the domain of an item from p.

The items are modeled by their *life cycle graphs*, which represent the state each item can be in during its life cycle in the system. An example of a life cycle graph can be seen in figure 1. States in this graph are possible states for the item such as *Available*, or *Reserved*. Transitions represent the effect of actions such as reserving an item or buying it.

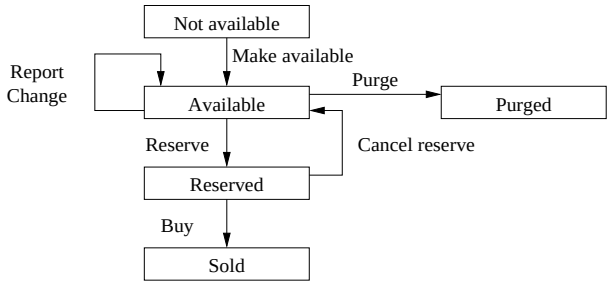

Fig. 1. The life cycle graph of product's item

Each item from I has several attributes, including the associated product, its state, and other characteristics. Finally, the agents are represented by concurrent processes that execute services, which are sequences of transitions on the state-transition graphs.

In this model, each global state represents one state in each product life cycle graph, and transitions model the effects of actions in the system. Therefore, paths in the global graph represent events that can occur in the system. The life cycle of the product is the set of all life cycles of its items.

3.3 Application Level

This level describes the e-commerce system in terms of the life cycle of the items. It is necessary to identify the states of an item, its attributes, the set of actions that could be executed on it and the effects caused by them and the agents that execute these actions. Here we are not interested in the functionalities of the web site and the architecture of them yet.

In this level it is important to verify the completeness property of the e-commerce model. Here, it is important to observe that there are only actions and states. Actions, by definition, are transactional, so the atomicity, consistency and isolation are guaranteed. Transitivity is related to the functionalities, so it will be important only in the next level, where there are services being executed in the model.

To check the completeness property of the business model, we use the CTL-formulas below, where S consists of all the states presented in the application model and A, the universe of actions.

```
EF (state = <S>)        EF (action = <A>)
```

In this level there are agents (Seller and Buyer) that represent the consumer and the supplier of the system. There is an item, which has a set of states. The agents execute actions that could affect the item's state.

3.4 Functional Level

This level introduces the product, composed by zero (the product is not available) or more items. The designer determines the operations the agents can perform, denoted as services. A service is executed on products and its effects might change or not the state of it and its items. The focus of this level is to define clearly how the services are executed and what happened with the product and its items in this case.

In this level it is important to verify the transitivity property of the model. The agents execute services that change the state of the item. This state must be consistent with the life cycle of the item and the related business rule associated with it. An example of transitivity is:

```
AG ((item_state=Not_Available & service=Make_Available) ->
AX (item_state=Available)
```

In this level it is important to verify the atomicity, consistency and isolation properties either. It is essential to check the consistency between the state of the product and its items in a given moment. Moreover, there are agents performing services concurrently, which may cause the system to achieve an invalid state. Therefore the isolation property must be guaranteed.

To become clear, we give some examples of this properties[2]. First we give an example of atomicity. if an item is available and a *Reserve* service is performed by a buyer agent and granted by the server, the item must be reserved in the next state and the inventory must be decremented.

```
AG ((item_state=Available & service=Reserve & product_inventory=1)
-> AX (item_state=Reserved & product_inventory=0))
```

An example of consistency property can be seen as follow:

- If the inventory is zero, then no item should be available.

```
AG (product_inventory = 0 -> !product_state = Available)
```

[2] The actual properties verified are slightly different than the ones presented here, which have been simplified for readability.

An example of isolation property could be: if there are two items available and two buyers reserve these items simultaneously, the inventory must be zero in the next step.

```
AG ((buyer1_service=Reserve & buyer2_service=Reserve
& product_inventory=2) -> AX (product_inventory=0))
```

In this level there are agents that execute services, that could modify the product's state. Some of these services, as a *Reserve* of an instance of the product, change the state of the item either.

It is important to notice that the properties validated in the second level should retain their validity in the third one and so on. The verification of the properties should be incremental as well as the methodology proposed.

3.5 Architectural Level

This level specifies the system in terms of its components and the way they interact with each other. It is important to emphasize that this level encompasses the other ones, completing the specification of the system and describing its architecture.

In this stage, the model is more complex, contemplating the components of the system's architecture. We identify the atomicity properties that should be verified and other consistency and isolation rules. The transactional properties verified in the other levels should be checked again, as the transitivity properties.

We introduce the components of the e-commerce system: the web server, the transaction server and the database. There are agents that submit requests to the web server, which translate them into operations to the transaction server. These operations, named services, are executed by this server, sometimes performing action on the items. This level is important because it enables the designer to get a specification closer to the real implementation he wants to develop.

4 Case Study: An E-commerce Virtual Store

In this section we will present our case study, an e-commerce virtual store, a very useful and popular application. Our goal is to show how the methodology proposed can be used to design more reliable systems. This is a typical electronic commerce application in which most of the aspects that make such applications complex to design are present, such as multiple agents of different types that compete for access to products, products with more than one item and intermediate states for items (for example, one buyer should reserve an item before buying it). We have used the SMV model checker [7] to perform this task.

4.1 Conceptual Level

In the virtual store we modeled, there are six states which correspond to types of pages on the web site: *Home*, *Browse*, *Search*, *Select*, *Add* and *Pay*. The *Home*

page is the initial web page of the site. The *Select* page shows specific information about a product. The *Add* page confirms product reservations and displays the contents of the customer's shop cart. The *Pay* page is loaded after the purchase of the items in the shop cart is completed. The *Search* and *Browse* pages present general information about the products offered by the virtual store. There are still some states that correspond to the administration view of the web site, used by the seller agent to change information about the products (operation *Change*) and modify the its inventory (operation *Make Available*).

The transitions between these pages are associated with actions executed by the agents. An example is the execution of a reserve action by the buyer agent, that causes the transition from *Select* to *Add* if completed with success. Therefore a transition between two web pages is mapped to an action in the life cycle graph of the product's item.

In our virtual store, we have modeled two types of agents. The *Buyer* Agent represents the customers that access the virtual store through WWW to get information about the product and potentially to buy it. The *Seller* Agent represents the product's supplier that will make it available, update its data and potentially sell it. The buyer agent can execute one of the following actions: *Report*, the client requests information about the product; *Reserve/Cancel Reserve*, the client reserves an item or cancel a previous reserve; and *Buy*. The seller agent can execute one of the actions: *Make Available*, when a new item enters the store; *Change* to change its attributes; and *Purge*, when the item is removed from the store.

The life cycle graph of the product's item can be seen in Figure 1. The global model of the virtual store is a collection of life's cycle graphs and additional attributes are represented by variables such as the inventory (the number of product items available). Additional logic is needed to "glue" together the various life cycle graphs. For example, if a reserve is requested for one item but several are available, the store must decide which item will be reserved.

Finally, the agents are modeled as concurrent processes that perform actions. In the model there is one seller agent that represents the administrator of the store and one or more buyer agents that act as the customers. To illustrate how the methodology works in practice, we will present parts of the SMV code for the virtual store.

As defined in the methodology, Section 3.2, conceptual level details the e-commerce system requirements. We list some of the business rules we had identified in our case study:

- If the item is in the state *Not Available* and the action *Make Available* occurs, the next state is *Available*;
- If the item is in the state *Reserved* and the action *Buy* occurs, the next state is *Sold*;
- If the inventory is positive, at least one item must be available;
- The actions *Reserve* and *Cancel Reserve* must be atomic;
- And if there are agents executing concurrently, their actions must be isolated.

4.2 Application Level

A module in SMV consists of a number of variables and their assignments. The main module consists of the parallel composition of each module. This is accomplished by instantiating each module in the main module shown as follow:

```
MODULE main

VAR ba1: buyer_agent();
    ba2: buyer_agent();
    sa1: seller_agent();
    it1: item();
```

As described in Section 3.3, the first important property to be verified is completeness. In this case study, we can do this through the specification written in CTL formulas:

```
EF (it1.state = Not_Available)
...
EF (it1.state = Purged)
EF (ba1.action = Buy)
EF (ba2.action = Buy)
EF (sa1.action = Make_Available)
```

These specifications should be consistent with the item's life cycle graph, as illustrated by Figure 1. In our model all of them were verified as *true*, certifying its completeness.

4.3 Functional Level

Continuing the process defined by the methodology we add new modules to the model, which represents the product and its items. Here, we are interested in verify some business rules related to services.

Initially, as described in Section 3.4, we have to check the transitivity properties of the model. We can perform this using the following CTL formulas:

```
AG(state = Available & service = Purge) -> AX(state = Purged)
...
AG(state = Available & service = Reserve) -> AX(state = Reserved)
AG(state = Reserved & service = Buy) -> AX(state = Sold)
```

Here, we explain some transactional properties, beginning with atomicity. if an item is available and a reserve action is performed by a buyer, the item must be reserved in the next state and the state must be consistent with this or the service is not executed and the state is not modified.

```
AG ((state = Available & service = Reserve & inventory = v) ->
AX ((state = Available & inventory = v) |
    (state = Reserved & inventory = v-1)))
```

Note that the variable *inventory* partakes of the proposition added to this formula to verify this business rule.

The next formulas illustrate some consistency properties of the virtual store modeled.

The inventory should not be negative:

```
AG !(inventory < 0)
```

If the inventory is positive, at least one item must be available:

```
AG ((inventory > 0) -> (product_state = Available))
```

Finally we present examples of isolation.

If there are two buyer agents, one reserving the item and the other canceling his/her reservation, the inventory must be kept consistent after the execution of both services:

```
AG ((buyer1_service = Reserve & buyer2_service = Cancel_Reserve &
inventory = v) -> AX (inventory = v))
```

In the case of *inventory = 0*, the reservation service can not be preceded by the cancellation service. So, to solve this problem we decide to give priority to the buyer agent that wants to cancel the reservation.

In a similar way, we specified all the other business rules and verify their veracity.

4.4 Architectural Level

In this stage we added new modules to represent the e-commerce system as real as possible. So we include the web server, transaction server and database server in the model, adapting the specifications to it. Thus, the properties are related to requests, instead of services.

In this level we do not identify new properties since all the business rules were verified in the previous level.

The complete model has more than 10^{23} states with more than 10^{14} reachable states, used about 17MB of memory, and verification has been completed in just nine minutes for all properties.

5 Related Work

Formal analysis and verification of electronic commerce systems have not been studied in detail until recently. Most work such as [6, 1, 3, 10] concentrates on verifying properties of specific protocols and do not address how these techniques can assist in the design of new systems. Moreover, these techniques seem to be less efficient than ours, ranging from theorem proving techniques [6, 1] which are traditionally less efficient (even though more expressive), to model checking [3, 10]. But even these works tend to be able to verify only smaller systems consuming much higher resources than our method.

6 Conclusions and Future Work

In this paper we propose a methodology to specify e-commerce systems. This technique can increase the efficiency of the design of electronic commerce applications. We use formal methods not only to formalize the specification of the system but also to automatically verify properties that it must satisfy. This technique can lead to more reliable, less expensive applications that are developed significantly faster. We have modeled and verified a virtual store to demonstrate how the method works. As a result of our case study, we were able to detect a serious error, that violated the isolation property, causing the same item to be sold twice. It occurred because two buyer agents tried to acquire the product at the same time and there was only one item available. During this verification we have precisely identified both errors and their causes that would have been difficult to find out otherwise.

The proposed method can be applied in general e-commerce systems, where the business rules can be modeled by state transitions of the items on sale. As the method is based on CTL-formulae, the business rules should be represented by them, what is considered a limitation of the method.

We are currently studying other features of electronic commerce systems that we have not yet formalized, as well as the possibility of generating the actual code that will implement the system from its specification.

References

1. Bolignano. Towards the formal verification of electronic commerce protocols. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop* (1997), IEEE Computer Society Press.
2. Clarke, E. M., Grumberg, O., and Peled, D. A. *Model Checking.* The MIT Press, Cambridge, Massachusetts, 1999.
3. Department, S. L. Model checking the secure electronic transaction (set) protocol. In *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (1998).
4. Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. Property specification patterns for finite-state verification. In *2nd Workshop on Formal Methods in Software Practice* (March 1998).
5. Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. Patterns in property specifications for finite-state verification. In *21st International Conference on Software Engineering* (May 1999).
6. Gurgens, S., Lopez, J., and Peralta, R. Efficient detection of failure modes in electronic commerce protocols. In *DEXA Workshop* (1999), pp. 850–857.
7. K.L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* PhD thesis, Pittsburgh, PA, 1992.
8. K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Norwell Massachusetts, 1993.
9. Meira Jr., W., Murta, C. D., Campos, S. V. A., and Neto, D. O. G. *Sistemas de Comercio Eletronico, Projeto e Desenvolvimento.* Campus, 2002.
10. Wang, W., Hidvégi, Z., Bailey, A., and Whinston, A. E-process design and assurance using model checking. In *IEEE Computer* (Oct. 2000).