Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

The ASSERT Virtual Machine: A Predictable Platform for Real-Time Systems ⋆

Juan A. de la Puente * **Juan Zamorano** *
José A. Pulido * **Santiago Urueña** *

* *Universidad Politécnica de Madrid (UPM), E-28040 Madrid, Spain*

Abstract: The development of real-time control systems is a complex process which has to face often conflicting requirements, especially those related to the performance of the control methods and the real-time behaviour of the system. The ASSERT Virtual Machine provides a reliable execution platform for such systems, which allows developers to cope with functional and real-time aspects separately. In order to guarantee the required real-time properties, the virtual machine only accepts software components which have a predictable temporal behaviour which can be analysed at system design time. Such components can be automatically generated from a high-level description of a system which embodies the functional components (e.g. control algorithms) into a set of containers providing the appropriate concurrent and real-time behaviour. The ASSERT Virtual Machine has been implemented in Ada 2005, using a predictable tasking subset of the language known as the Ravenscar profile. A prototype has been validated on several pilot-scale spacecraft control systems, with good results.

Keywords: Computers for control, real-time systems, programming environments, software engineering, model-driven development.

1. INTRODUCTION

Real-time control systems have complex, and often conflicting requirements. On one side, control algorithms have to be designed and tuned for the required performance metrics. On the other side, the execution of control algorithms on the chosen computer platform must exhibit an appropriate temporal behaviour, e.g. with respect to periodic execution, limited jitter, etc. Additional requirements may refer to dependabilty or security properties, power consumption, or overall cost. This complexity often makes the development process of such systems extremely difficult to manage, and results in added cost and long development times.

A common approach to real-time system development is based on the concept of *separation of concerns*. Control algorithms and other functional parts of the software are developed from scratch or, more often, with the help of some model-based tool such as *Simulink*. Support for concurrency and real-time is added at a later stage, usually by hand, using a cyclic executive or better a real-time kernel [see e.g. Liu 2000]. Although there are indeed interactions between the design of control algorithms and the real-time aspects of the system [Albertos and Crespo, 2001, Cervin et al., 2003], in many cases the separation approach provides a convenient way to ensure the desired control and real-time performance in an industrial framework. Its main problem is the difficulty of decoupling the functional algorithms from the design of concurrent and real-time features of the system.

The ASSERT [1] project is aimed at developing enhanced development processes for a particular kind of real-time control systems, embedded on-board aerospace systems. The project has been carried out by a consortium of 29 industrial and academic partners led by the European Space Agency (ESA) [2]. Since the application domain is representative of control systems with hard real-time and high-integrity requirements, its results may be expected to be applicable to other domains as well. The approach adopted relies on the development of a set of *building blocks* which can be used in open frameworks in order to develop software for *system families*, i.e. sets of related system products with a common architecture. Software component models and formal methods are used when appropriate in order to verify the correctness of the software. To this purpose, a new software development process has been devised, based on the separation principle: functional code is embedded into *containers* which provide the required concurrency and timing elements (e.g. threads and synchronization mechanisms). Containers undergo a series of transformations until they are in a form that can be directly executed on a specialised platform, the ASSERT *Virtual Machine*, which guarantees that the behaviour of the synchronization and timing mechanisms is correct.

⋆ This work has been partially funded by the Spanish Ministry of Education, project no. TIC2005-08665-C03-01 (THREAD), and by the IST Programme of the European Commission under project IST-004033 (ASSERT).

[1] *Automated proof-based System and Software Engineering for Real-Time systems.*
[2] `http://www.assert-project.net`

This paper focuses on the concept and the characteristics of the ASSERT Virtual Machine (VM), especially those related to real-time performance. The general framework provided by the ASSERT development process is introduced in section 2. The design of the Virtual Machine is described in detail in section 3. Section 4 contains a description of the VM implementation, and section 5 describes its validation on a pilot industrial project. Finally, conclusions of the work, as well as some ideas for future work are given in section 6.

2. THE ASSERT DEVELOPMENT PROCESS

The ASSERT development process for embedded real-time systems [Vardanega, 2006, Bordin and Vardanega, 2007] is a model-driven process based on the following principles:

- Definition of a specific meta-model that enables the relevant properties (e.g. timing parameters) to be included in system models;
- separation between functional ad concurrency and real-time aspects in system models;
- formal definition of properties and model transformations guaranteeing the preservation of the temporal behaviour of the system.

The meta-model provides a precise definition of the elements that can be used to build models of computer systems in a specific domain [OMG, 2003]. The ASSERT meta-model is a formalization of the Ada Ravenscar Profile [Burns et al., 1998], a subset of the Ada concurrency model [ISO/IEC, 2007] for high-integrity real-time systems exhibiting a predictable, analysable time behaviour. This meta-model is called the *Ravenscar computational model* (RCM), and its main elements are [ISO/IEC, 2005]:

- A static set of periodic and sporadic concurrent tasks, with a bounded and known execution time.
- Communication between tasks by means of static shared objects with mutually exclusive access.
- Fixed priority scheduling of tasks, with immediate ceiling priority inheritance access to shared objects [Sha et al., 1990]

Separation between functional and temporal aspects is implemented by *containers*. A container is a generic component in which functional code is embedded in such a way that non-functional properties, especially real-time properties, are guaranteed by construction. Two different kinds of containers are used in ASSERT, with two corresponding levels of abstraction (figure 1):

- *Application-level containers* (APLC) are used to define the components of a system (e.g. sensors or controllers), and the interfaces between them, in an abstract way. Interfaces provide the functional interaction between components, i.e. the operations or methods that can be performed on a component (*provided interfaces*), or that a component may require from others (*required interfaces*). Non-functional properties are defined by means of a specially-defined interface grammar [Bordin and Vardanega, 2007].
- *Virtual-machine-level containers* (VMLC) define the components that are directly executed on top of the execution platform, i.e. the ASSERT Virtual Machine.Examples of VMLC include periodic and sporadic tasks, and shared data objects, which are instances of the Ravenscar Computational Model.

Containers provide support for concurrency, synchronization, and real-time to the embedded functional code, letting control systems designers concentrate on the design of control algorithms and specify temporal properties only at a high level of abstraction. For example, the functional code for a PID algorithm can be embedded in an AP-level container with a periodic behaviour specification, including a deadline in the production of the control action in order to limit the effect of output jitter. The APLC can then be transformed into one or more VM-level containers, depending on how it interfaces with other components. One of the resulting VMLC will provide a periodic task ensuring that the control algorithm is executed with the requested period, and that it finishes within deadline in each period. Ada code for such a task can then be automatically generated.

An important characteristic of this scheme is that the semantics of all components is based on a single meta-model, the RCM. This formal basis enables models to be transformed into each other while still preserving their real-time properties. Model transformations are the basic element of the ASSERT software development process, in which higher-level models (functional and interface models, based on APLC) are transformed to lower-level models (based on VMLC), which are directly executed on the ASSERT VM [Panunzio and Vardanega, 2007].

3. THE ASSERT VIRTUAL MACHINE

3.1 Run-time entities

The ASSERT Virtual Machine (VM) is the execution platform for the run-time components of a real-time system. In order to guarantee that the real-time properties which are specified in the upper abstraction levels are preserved, the VM only accepts entities which are "legal" with respect to the computational model, i.e. the RCM. These entities are built from a reduced set of *archetypes*:

- *Active components*: periodic and sporadic tasks. Periodic tasks execute an action which repeats every period, e.g. a control algorithm. Sporadic tasks execute an action whenever an *activation event* occurs, e.g. a change of settings.
- *Protected components* contain data objects that are shared by two or more active components, and are only accessible by means of a specific set of operations or methods. In order to protect shared objects from being corrupted by concurrent access, mutual exclusion is enforced on the execution of their operations.
- *Passive components*, which contain data object that are only used by a single active component. Passive objects are only accessible trough a set of operation which are part of the object definition.

Ada code running on top of the virtual machine can be automatically generated in such a way that conformity with the computational model is preserved. For such purpose, as above stated, functional code is embedded into

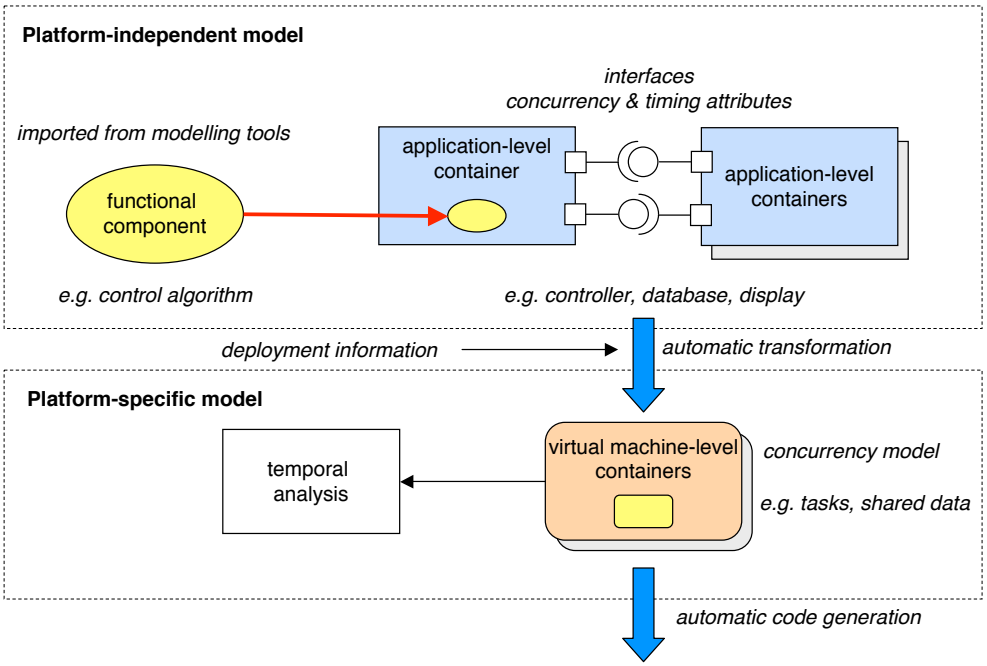

Fig. 1. ASSERT component model.

VM-level containers,[3] built from the previously defined archetypes, which provide mechanisms for encapsulating the associated temporal attributes (period, worst-case execution time, deadline, etc.). Containers may also include monitoring mechanisms for detecting possible timing errors occurring at run-time, so that corrective actions may be taken whenever possible. As an example, listing 1 shows a simplified archetype of a periodic container performing a control action, possibly imported from a model-based design tool. Listing 2 shows a slightly more complex container archetype, also including execution-time monitoring in order to detect possible WCET overruns. If such an error occurs, a procedure (Overrun_Hander in listing 2) is automatically started, which can be used to perform some appropriate error-correcting action, e.g. a system reconfiguration.

3.2 Functionality of the Virtual Machine

The main characteristic of the ASSERT Virtual Machine is that it provides run-time support for preserving the temporal properties that have been specified for the system components built from the basic VMLC archetypes. Since the underlying model for these archetypes is the Ravenscar Computational Model, the central component of the Virtual Machine is a Ravenscar-compliant real-time kernel, i.e. one that provides basic functionality for the execution of a static set of concurrent tasks with the following characteristics:

- Periodic activation of periodic tasks;
- event-based activation of sporadic tasks, enforcing a minimum separation between two consecutive activations;

[3] Embedded functional code does not have to be written in Ada. Any language that can be interfaced with Ada (e.g. C, C++ or even Fortran) can be used.

Listing 1. Periodic task.

```
task body Periodic_Task is
  Next_Activation : Ada.Real_Time.Time := Epoch;
begin
  loop
    delay until Next_Activation;
    Control_Action;   -- imported functional code
    Next_Activation :=
      Next_Activation + My_Period;
  end loop;
end Periodic_Task;
```

Listing 2. Periodic task with WCET overrun detection.

```
My_Identity : aliased constant Task_Id :=
   Periodic_Task'Identity;
WCET_Timer  : Ada.Execution_Time.Timers.Timer
  (My_Identity'Access);

task body Periodic_Task is
  Next_Activation : Ada.Real_Time.Time := Epoch;
begin
  loop
     WCET_Timer.Set_Handler
       (In_Time =>
          My_WCET_Budget,
        Handler =>
          My_Monitor.Overrun_Handler'Access);
    delay until Next_Activation;
    Control_Action;   -- imported functional code
    Next_Activation :=
      Next_Activation + My_Period;
  end loop;
end Periodic_Task;
```

- mutual exclusion in the execution of protected object operations;
- scheduling the execution of periodic and sporadic tasks in a predictable way, so that all tasks deadlines can be guaranteed whenever possible; [4]
- monitoring the processor time used by a task and detect violations of the specified worst-case execution-time.

By providing such basic functionality, the VM real-time kernel only accepts the execution of legal run-time components as defined in 3.1 above. Furthermore, if the kernel is used in conjunction with an Ada 2005 compiler, the profiling facilities of the language [ISO/IEC, 2007, app. D.13] can be used to force the compiler to accept only Ravenscar-compliant Ada programs, thus making sure that the temporal behaviour of the source code can be analysed with respect to the model requirements.

3.3 Architecture of the Virtual Machine

Although this paper concentrates on non-distributed systems, the ASSERT VM also supports distributed execution on a set of computer nodes linked by a predictable communication network. To this purpose, a hierarchical architecture has been designed that includes additional subsystems for communication and distribution (figure 2). The main elements of the full virtual machine are:

- A real-time kernel supporting the Ravenscar computational model (see 3.2 above).
- A set of communication drivers and protocols ensuring bounded transmission times. The SOIS standard for spacecraft communications [CCSDS, 2006] has been adopted as a basis for this subsystem.
- A middleware layer providing distribution transparency and reliable replication schemes for critical software components.

As above stated, the real-time kernel is the critical component for guaranteeing the real-time behaviour of a system. The next section describes the main features of its design and implementation.

4. IMPLEMENTATION

The choice of the Ravenscar Computational Model in ASSERT immediately suggests using Ada as the implementation language of the ASSERT VM. The Ravenscar profile is part of the last Ada 2005 standard [ISO/IEC, 2007], which also includes a complete set of timing and monitoring mechanisms that can be used to implement the above requirements [Pulido et al., 2007]. An instance of the ASSERT VM has been built based on ORK+, an evolved version of the Open Ravenscar Real-Time Kernel (ORK), which was developed by UPM as a dedicated operating system platform for spacecraft on-board systems [de la Puente et al., 2000, Zamorano and Ruiz, 2003]. The kernel is targeted to LEON2, a radiation-hardened implementation of the SPARC V8 architecture [SPARC International, 1992], and is integrated with a development

[4] The RCM specifies fixed-priority pre-emptive scheduling with ceiling-locking (i.e. immediate ceiling priority inheritance) access to protected objects.

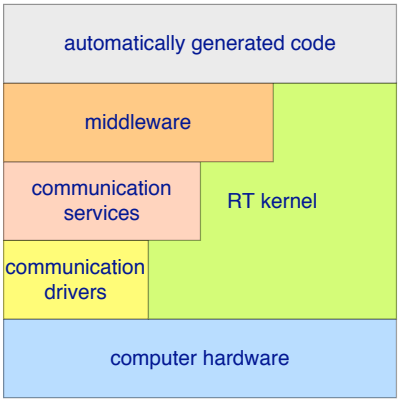

Fig. 2. ASSERT Virtual Machine Architecture.

version of GNAT for LEON, an industrial-grade Ada 2005 compilation system. It fully supports the Ada Ravenscar profile and the execution-time monitoring mechanisms.

The current version of the virtual machine also includes communication drivers and a protocol stack for the SpaceWire bus [ECS, 2003] and the SOIS message transfer service (MTS)[CCSDS, 2006], as well a high-integrity version of the PolyORB middleware layer [Hugues et al., 2006], called PolyORB-HI. The communication subsystem has been implemented in C, and compliance with the Ravenscar computational model has been provided by rigorous coding restrictions. The PolyORB-HI middleware is written in Ada with high-integrity restrictions [ISO/IEC, 2000], including the Ravenscar profile for tasking.

5. VALIDATION OF THE APPROACH

In order to validate the methods and tools developed in the ASSERT project, three *pilot projects* have been defined, lead by the main industrial partners of the project. The pilot projects are based on previous industrial developments, and focus on different aspects of spacecraft control systems. The projects are:

- HRI (Highly Reliable Infrastructure) is oriented at long-duration satellites with no possibility of maintenance;
- MPC (Multiple-Platforms and Cooperation) is oriented at distributed systems and satellite fleets;
- MA3S (Multi-domain Advanced Available Automated System) is centred on mission-critical control systems embedded in unmanned spacecraft.

The ASSERT VM has been used to support the development and execution of real-time software in the HRI and MPC projects, led respectively by Thales Alenia Space and Astrium Satellites. The HRI software includes typical spacecraft control subsystems, such as guidance and navigation control (GNC), battery management, thermal control, and telemetry-telecommand (TMTC). It also includes a couple of payload applications, an altimeter and an image processing application.

The functional code for the HRI system has been developed using a variety of modelling tools, including UML and Matlab/Simulink, targeting different programming

languages such as C, C++ and Ada. The code has been embedded into a number of application-level containers, from which a set of virtual-machine level containers has been derived using automatic transformation tools developed within the project. The generated Ada code runs on an embedded computer platform with two LEON2 computers connected by a SpaceWire network, each of them hosting an instance of the ASSERT Virtual Machine. The demonstration prototype also includes some additional computers which simulate a ground station and provide debugging and execution control utilities.

The MPC demonstrator simulates a constellation of three observation satellites. It also includes some real building blocks, such as attitude and orbit control (AOCS) and communications, as well as advanced coordination management modules. The system has also been developed with the help of transformation tools developed in the project, and the resulting code runs on a distributed system built with LEON2 computers as SpaceWire links, as before. The ASSERT virtual machine has been used as the run-time platform on all the system nodes.

The results of these experiments are encouraging and show that the code implementing the concurrency, synchronisation and real-time aspects of the system can be automatically generated from a high-level description embodying the control code coming from a model-based simulation and design environment. The resulting software has been exhaustively tested, showing that the specified real-time behaviour is indeed ensured by the Virtual Machine. Similar results have been obtained from a third experiment performed at ESA/ESTEC, [5] in which software for a typical spacecraft platform has been developed and run on a LEON2 simulator.

6. CONCLUSIONS

The approach to real-time software development described in this paper is based on the following principles:

- Separation between functional (control algorithms) and synchronization and timing aspects of a system;
- definition of a formal computation model based on well-known principles that ensure a predictable, analysable timing behaviour;
- software development process based on models and model transformations preserving the real-time properties of the system;
- software execution on top of a spetialized platform, the ASSERT Virtual Machine, that guarantees the real-time properties at run time and detects any possible violations of it.

This approach has shown its feasibility by the build of an instance of the ASSERT Virtual Machine, which has been successfully used in several industrial-grade projects involving sophisticated spacecraft control functions. Functional code developed with modelling tools commonly used in the spatial control systems domain, such as Matlab/Simulink and UML, has been seamlessly integrated into so-called containers, which embody all the concurrency, synchronization, and real-time complexity.

[5] European Space Research and Technology Centre.

Some recent additions to the ASSERT Virtual Machine which are still being assessed include:

- Logical partitions, enabling several applications to run on the same computer platform in such a way that faults arising in one of them do not propagate to the rest of the system.
- Full integration with the Ada 2005 object-oriented programming model;

A basic set of graphical tools supporting model development and transformation, as well as automatic code generation for systems running on the ASSERT Virtual Machine, has been built by other project partners. Most of the tools are integrated into well-known environments such as Eclipse, [6] Stood, [7] , and TOPCASED. [8]

ACKNOWLEDGEMENTS

The authors wish to acknowledge the contribution of José Redondo and Jorge López, from UPM, to the implementation of the ASSERT VM prototype.

The fundamental ideas on containers and transformations originated at the University of Padua, by Tullio Vardanega, Marco Panunzio, and Matteo Bordin.

The work on the PolyORB-HI middleware has been carried out at ENST Paris, under the direction of Laurent Pautet and Jérôme Hugues.

The communication drivers and protocols used in the first VM prototype have been developed by Stuart Fowell and Marek Prochazka, from SciSys.

The authors would also like to acknowledge the fruitful collaboration with the rest of the partners of the ASSERT project, and especially Mathieu Le Coroller, Gérald Garcia and Guy Estaves from Thales Alenia Space, Dave Thomas from Astrium Satellites, and Maxime Perrotin from ESA/ESTEC, in building the final demonstrators.

REFERENCES

Pedro Albertos and Alfons Crespo. Integrated design and implementation of digital controllers. *Lecture Notes in Computer Science*, 30:385–392, 2001. ISSN 540-42959-X/0302-9743.

Matteo Bordin and Tullio Vardanega. Correctness by construction for high-integrity real-time systems: A metamodel-driven approach. In Nabil Abdennadher and Fabrice Kordon, editors, *12th International Conference on Reliable Software Technologies — Ada-Europe 2007*, number 4498 in LNCS, pages 114–127. Springer-Verlag, 2007.

Alan Burns, Brian Dobbing, and George Romanski. The Ravenscar tasking profile for high integrity real-time programs. In Lars Asplund, editor, *Reliable Software Technologies — Ada-Europe'98*, number 1411 in LNCS, pages 263–275. Springer-Verlag, 1998.

CCSDS. *Spacecraft On-board Interface Services Green Book — CCSDS 850.0-G-0.1.* Consultative Committee for Space Data Standards, November 2006. Draft.

[6] `http://www.eclipse.org`
[7] `http://www.ellidiss.com`
[8] `http://www.topcased.org`

Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and Karl-Erik Årzén. How does control timing affect performance? *IEEE Control Systems Magazine*, 23(3): 16–30, June 2003.

Juan A. de la Puente, José F. Ruiz, and Juan Zamorano. An open Ravenscar real-time kernel for GNAT. In Hubert B. Keller and Erhard Plödereder, editors, *Reliable Software Technologies — Ada-Europe 2000*, number 1845 in LNCS, pages 5–15. Springer-Verlag, 2000.

ECSS-E-50-12A: Space engineering — SpaceWire — Links, nodes, routers and networks. ECSS, January 2003. Available from ESA.

Jérôme Hugues, Bechir Zalila, and Laurent Pautet. Middleware and tool suite for high integrity systems. In *Proceedings of RTSS-WiP'06*, pages 1–4, Rio de Janeiro, Brazil, December 2006. IEEE.

ISO/IEC. *Ada 2005 Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652:1995/Amd 1*, 2007. Published by Springer-Verlag, ISBN 978-3-540-69335-2.

ISO/IEC. *TR 15942:2000 — Guide for the use of the Ada programming language in high integrity systems*, 2000.

ISO/IEC. *TR 24718:2005 — Guide for the use of the Ada Ravenscar Profile in high integrity systems*, 2005. Based on the University of York Technical Report YCS-2003-348 (2003).

Jane W. S. Liu. *Real-Time Systems.* Prentice-Hall, 2000.

OMG. *MDA Guide Version 1.0.1*, 2003. Available at `http://www.omg.org/mda/`.

Marco Panunzio and Tullio Vardanega. A metamodel-driven process featuring advanced model-based timing analysis. In Nabil Abdennadher and Fabrice Kordon, editors, *12th International Conference on Reliable Software Technologies — Ada-Europe 2007*, number 4498 in LNCS, pages 128–141. Springer-Verlag, 2007.

José A. Pulido, Santiago Urueña, Juan Zamorano, and Juan A. de la Puente. Handling temporal faults in Ada 2005. In Nabil Abdennadher and Fabrice Kordon, editors, *Reliable Software Technologies — Ada-Europe 2007*, number 4498 in LNCS, pages 15–28. Springer-Verlag, 2007.

Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Tr. on Computers*, 39(9), 1990.

SPARC International. *The SPARC architecture manual: Version 8.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992. ISBN 0-13-825001-4. URL `http://www.sparc.com/standards/V8.pdf`.

Tullio Vardanega. A property-preserving reuse-geared approach to model-driven development. In *Proceedings. 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2006*, pages 223–232, August 2006.

Juan Zamorano and José F. Ruiz. GNAT/ORK: An open cross-development environment for embedded Ravenscar-Ada software. In Eduardo F. Camacho, Luis Basañez, and Juan Antonio de la Puente, editors, *Proceedings of the 15th IFAC World Congress.* Elsevier Press, 2003. ISBN 0-08-044184-X.