Automatic Generation of Run-Time Monitoring Capabilities to Petri Nets Based Controllers with Graphical User Interfaces

Fernando Pereira[1,2], Luis Gomes[1,3], and Filipe Moutinho[1,3]

[1] FCT/UNL Universidade Nova de Lisboa
[2] ISEL Instituto Superior de Engenharia de Lisboa
[3] UNINOVA, Portugal
fjp@deea.isel.ipl.pt, lugo@fct.unl.pt, fcm@uninova.pt

Abstract. The growing processing power available in FPGAs and other embedded platforms, associated with the ability to generate high resolution images and interface with pointing devices, opened the possibility to create devices with sophisticated user interfaces. This paper presents an innovative tool to automatically generate debug, diagnostic and monitoring graphical interfaces to be integrated in embedded systems designed using Petri net based controllers. Devices powered with the new debug and diagnostic interfaces benefit from lower maintenance costs and simplified failure diagnostic capabilities, leading to longer product life cycles with the corresponding environmental and sustainability gains. To demonstrate the validity of the tools proposed, the paper presents an application example for a Car Parking controller, including results on a working prototype.

Keywords: Embedded Systems, Petri nets, Design automation, Modeling, Graphical User Interfaces.

1 Introduction

Graphical debug and monitoring tools have always played a very important role in the development of embedded and automation systems. Due to the lack of resources and processing power available in hardware used to deploy these solutions, the tools have traditionally relied on software running on external personal computers.

However, the growing adoption of reconfigurable hardware platforms (ex. FPGAs) in embedded and industrial automation solutions brought increased processing power associated with the capability to generate high resolution images and interface with pointing devices, as mice and touch-screens, with no significant additional cost.

This paper presents a tool framework to the automatic generation of graphical debug, diagnostic and monitoring interfaces directly in FPGA hardware. This approach has many advantages over traditional solutions because the tools can be used after the development, test and validation phase terminates, to perform maintenance tasks and help diagnose mechanical and electrical faults.

The new tool takes advantage and extends a previous framework [1] containing design and modeling tools based on IOPT (Input-Output Place-Transition) Petri nets

Luis M. Camarinha-Matos (Ed.): DoCEIS 2011, IFIP AICT 349, pp. 246–255, 2011.

[2], simulation and automatic code generation tools for micro-controllers or FPGAs, plus an Animator tool to produce Graphical User Interfaces associated with the IOPT model execution [3].

The proposed solution analyzes a PNML file [4] generated by the referred tool chain describing an embedded system controller and automatically creates a set of XML files to the Animator tool, containing a debug and monitoring animation screen. This screen contains a graphical image of the IOPT Petri net model and a set of animation rules to display the status of the model in real time, including net marking, transition status and input and output signals. The system designer can later integrate this animation screen in the final application user interface.

2 Contribution to Technological Innovation and Sustainability

The main innovation presented in this paper is the capability to automatically generate debug and monitoring graphical interfaces for embedded systems, with zero additional design effort and negligible cost. The complete tool chain can generate full embedded system controllers for FPGAs, including the controller and an animated GUI with a debug and monitoring interface, without writing a single line of code.

Adding graphical debug and monitoring interfaces to embedded devices can have an enormous environmental impact and greatly contribute for sustainability. To better understand those impacts, embedded devices should be separated into two classes: industrial automation systems and end-user appliances.

Industrial automation systems generally have high availability requirements because downtime in one system can stall entire production lines, causing effective downtime over entire production plants, with the consequent delivery delays and high labour loss. In light of this problem, the performance of maintenance and technical assistance services is regarded with special importance.

Technical assistance interventions are generally characterized by a typical pattern: whenever an assistance call is received by an equipment supplier, a technical team is immediately scheduled to visit the costumer's site and diagnose the problem, returning home to fetch the required parts, followed by a second visit to implement a solution.

Embedding diagnostic and monitoring capabilities in the final systems can effectively break this pattern, as machine operators and the factory's maintenance engineers, with the help of the vendor's remote assistance, have the means to diagnose problems and identify damaged parts, reducing the number of travels to just one. Factory's maintenance engineers can even receive training to use auto-diagnostic systems to solve most problems and replacement parts can be sent using express mail services, avoiding the need to send technicians altogether. This solution results in faster repair times, minimized down-times and equipment suppliers can operate with smaller technical assistance vehicle fleets, reducing energy consumption and contributing for sustainability.

Another indirect result is the reduction of redundant production units: to minimize downtime, industrial facilities generally purchase spare units of the most sensitive machinery. The number of redundant units is calculated according to past failure statistics (MTBF) and average repair time. Lower average repair times enable the reduction of spare units, contributing even more to sustainability.

Embedded systems present in end-user appliances can also benefit from internal diagnostic and monitoring interfaces. In this class of systems, a high percentage of technical assistance incidents is related to improper user operation and bad device configuration, as users did not receive adequate training, resulting in unnecessary travels to support centers. The addition of internal diagnostic graphical interfaces provide an effective way to simplify the communication between end-users and help-desk staff, allowing to solve most problems remotely.

When devices suffer from real defects, internal debug and diagnostic interfaces can provide the same gains experienced by industrial systems: end-users and help-desk staff can cooperate, diagnosing failures remotely and making possible to send a technician with all the necessary parts to quickly solve the problem.

Due to the lack of capability to diagnose and solve problems in a single visit, most brand-name manufacturers strategy consists in immediately replacing systems covered by warranty with new units. This approach poses many environmental hazards, as the consumables present in the old systems are simply disposed as garbage and the old systems cannot be resold as new after repair, being often also disposed.

When appliances malfunction after warranties expire, there is a common perception that they are not worth repair, due to the low cost of new units, high transportation costs and high technical-center fees. This happens even when faults are caused by trivial problems as a loosen screw, a melted fuse or a wrong EPROM configuration.

The addition of integrated debug and diagnostic interfaces can help users detect most problems and repair the most trivial ones or use the service of local repair shops. These shops used to be very popular several decades ago, but the advent of ever increasing complex electronic devices, requiring the use of specialized diagnostic equipment, turned the repair of sophisticated electronic devices almost impossible. The addition of integrated diagnostic and debug interfaces can revive local repair shops and allow end users repair trivial problems, largely increasing the useful life cycle of consumer devices, minimizing the creation of hazardous garbage and contributing to great environmental and sustainability gains.

3 Comparison with Present Solutions

Debug and monitoring interfaces present in embedded systems development tools generally run on external computers connected to the physical embedded systems using special purpose data cables. This is the usual method employed in industrial programmable logic controllers. However, an industrial facility generally contains many systems from multiple vendors and it is not always possible to hold the development tools for all of them.

Even the development tools may not be enough to run diagnostics because the tools often require access to the real model files used during system development. However, equipment suppliers may deny access to the development model files to hide implementation secrets and to prevent unauthorized changes that may compromise safety and regulation compliance, leading to possible legal problems.

On the contrary, the diagnostic interfaces produced by the new tools are available to end users without requiring any dedicated hardware or software, yet do not allow

system changes. To prevent access to implementation secrets, vendors can choose to install a simplified debug model, providing enough information to diagnose failures, but hiding sensitive information.

Most current end-user appliances include some degree of self-test and diagnostic utilities. For example, some printers have self-test pages and many devices generate error codes presented as display messages or blinking LED counts.

However, the self-test routines must be specifically programmed during the development phase and only detect typical failures predicted by system developers. On the contrary, the new tools do not require programming and automatically append a debug and diagnostic interface to the final system, helping diagnose all types of failures, including those not originally foreseen.

More importantly, the traditional diagnostic routines do not work when a device reaches a deadlock situation. To perform diagnostics the device must be restarted, failing to identify transient error conditions. In alternative, the new diagnostic interfaces run in parallel with the system controller and can be recalled at any time, independently of the main controller status and without causing any state changes, thus allowing the detection of deadlock and transient faults.

Finally, the error codes generated by current devices often have no value to the end users because the meaning of the codes is only available to manufacturer's technical staff. On the opposite, the new debug and diagnostic interfaces provide an intuitive animated graphical user interface, displaying the state of the system in real time as a Petri net model, which is the underlying modeling formalism.

4 Related Work

Embedded systems design based on Petri net models has been the subject of many research publications, ranging from Low level nets [2][5] to High level colored nets [6][7][8].

Most development frameworks based on Petri net models include debug and visualization interfaces [8][9], to exhibit the system state, but these tools generally work only during simulations running on personal computers and have not been ported to physical embedded devices.

Some Petri net frameworks include tools to create interactive animations [8][10]. For instance, the Colored Petri net Tools, a very successful modeling tool-chain, contains animation design tools [11][12] to enhance user-friendliness and simplify communication with persons with no knowledge about Petri net formalisms.

Other authors have worked on automatic code generation from Petri net models [13][14][15], to allow the rapid development of embedded applications. These tools automatically generate software source code and low level hardware descriptions implementing the behavior described by the model.

The contribution presented in this paper is based on a previous work [1][3] combining automatic software and hardware co-generation with the capability to design animated graphical user interfaces for embedded systems. The animations are automatically generated from the original Petri net model and a set of rules associating the visibility and position of graphical objects, varying according to the system state evolution.

Using the previous tools, a designer could rapidly create embedded applications with sophisticated graphical user interfaces, without writing code and without needing deep understanding about software and hardware design, thus bringing embedded systems design to a much broader audience.

In fact, the previous generation of tools already included the capability to implement debug and diagnostic interfaces, but those interfaces had to be designed by human operators, drawing a background image and defining a large set of rules to display the system state, one at a time. However, this operation was repetitive, error prone and time consuming, specially with large models. In contrast, the new tool generates the desired results without any human intervention.

5 Development Flow

The tools introduced in this paper are based on a development flow presented in [16], beginning with the analysis of system behavior through the identification of patterns of use, leading to the creation of UML Use-Case diagrams. The captured use-cases will then be modeled using IOPT Petri nets [2], with the help of the Snoopy IOPT net editor. After a first version of the controller model is finished, the editor will generate a PNML [4] file, with the corresponding XML representation. This PNML file will be the base for all subsequent development steps.

IOPT nets are a non-autonomous Petri net class inheriting the characteristics from the well known Place-Transition nets [17], with the capability to associate input and output signals and events to model elements, enabling the deterministic specification of the interaction between models and the external environment. IOPTs also benefit from maximal step semantics, meaning that all autonomously enabled transitions will immediately fire as soon as the associated guard conditions and events are active. To enable automatic conflict resolution, IOPT nets include transition priorities and other characteristics, like test arcs and arc weights that improve modeling capabilities.

Automatic code generation is performed through two applications, PNML2VHDL and PNML2C, that create system controllers based on VHDL hardware descriptions or “C” source code, according to the desired embedded target device.

Using the PNML file as input to an “Animator” tool, the developer can create a graphical user interface, consisting of multiple screens containing animated graphical objects. The PNML model is used as a base to define a set of rules describing the evolution of the animated graphical objects.

The animations created in the previous step can be presented on a personal computer during model simulations, to debug, validate and correct the designed controller model. Another tool – GUIGen4FPGA - will automatically generate VHDL code and EEPROM image files, to enable the execution on FPGAs.

To finally generate a running embedded application it is necessary to configure the physical hardware platform, through the association of model signal names and events to real FPGA pins, using a Xilinx UCF file.

The proposed development flow includes a simulation and animation step where the user can test and correct the designed model, returning to the first development stage whenever incorrect behavior is detected. However, the validity of the simulation phase largely depends on the ability to also model and simulate the physical environment surrounding the embedded controller, which can be a complex task. In those cases the validity of the models can only be checked on a physical prototype.

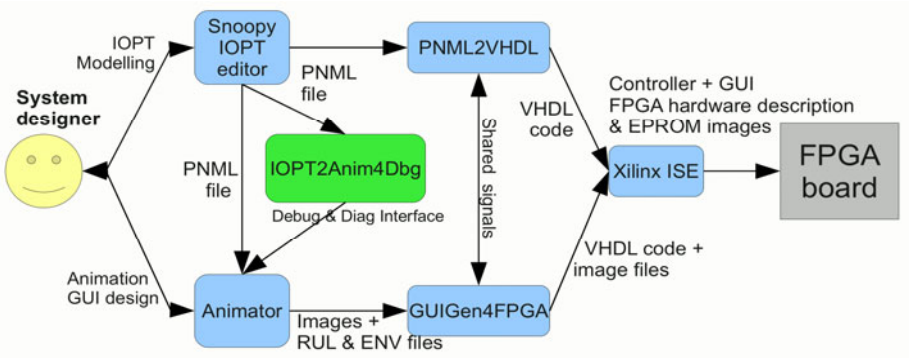

Fig. 1. Proposed development framework

One of the goals of the present work is to provide tools to help debug and identify mistakes during the prototype test phase. This way, the former development flow has been improved and a new tool "PNML2Anim4Dbg" was developed to automatically generate debug and diagnostic animation screens. Figure 1 displays a diagram describing the improved development tool chain.

6 Implementation

The tool introduced in this work, "PNML2Anim4Dbg", receives input from a PNML file containing a IOPT Petri net model describing a system controller and automatically generates a set of XML files for the "Animator" tool.

As both the input and output files are encoded using XML, the XSLT (Extensible Style-sheet Transformation) [18] framework was selected to implement the new tool, due to the capability to automatically validate syntactic grammars using DTDs or Schema and XML pattern matching, XML tree navigation and query tools (XPATH).

The usage of XSL transformations applied to PNML files is as old as the PNML format itself, since the first documents introducing the PNML standard [4], already proposed XSL transformations as a tool to convert models between different Petri net classes.

As seen in figure 2, the PNML2Anim4Dbg creates 5 files: A SVG background image, a «rul» file containing all generated animation rules, an «env» file associating a list of BMP image files to shortcut names present in the animation rules, an «ov» file with a list of output values generated by the controller and a «pdc» file containing a list of input signals and the corresponding user interface methods.

Although other XSLT based PNML to SVG converters were available [19], a new IOPT2SVG transformation was created, to account with the non-autonomous nature of IOPTs, including input and output signal graphical representations, different test and normal arc representations, transition priorities, etc.

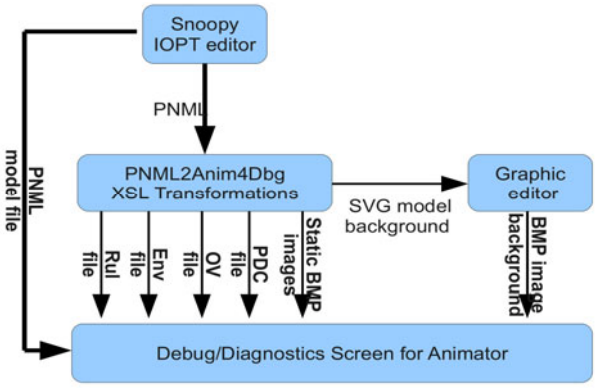

Fig. 2. PNML2Anin4Dbg data flow

The SVG background file contains an exact image of the IOPT model and can be edited and rearranged using most vector graphics editors, and finally converted to BMP format. During this phase it is possible to hide model comments and other superfluous information. Together with the SVG background image, a set of static image files is also appended to the animation project, containing pictures to display tokens inside places, signs to highlight the autonomously enabled transitions and LED signs to display active input and output signals.

The rules file contains a large set of rules to draw the correct number of tokens inside each place, to check if each transition is autonomously enabled and to check for active I/O signals. For more information about implementation details, the complete source code will be available online.

Complex hierarchical models, composed of several sub-net components, can be processed at different abstraction levels. Developers can choose simplified models showing only the top level components, full detailed models describing the entire system in a flat network, or create an hierarchy of multiple interface screens showing individual components. The tools do not require the entire model and can work with partial models, just requiring identifier consistency with the final controller system, maintaining the same place, transition and signal names.

7 Test and Validation

To test and validate the new tools, a car parking lot controller IOPT model (fig. 3) was processed using the proposed development flow to automatically generate a working prototype with a debug and diagnostic interface. This model is simple enough to eliminate the need for a detailed explanation, yet allows the demonstration of the proposed tools.

The hardware prototype was implemented using a Xilinx Spartan 3A 1800 Video Kit, including a Spartan 3A-DSP 1800 Starter Board, an Avnet EXP PS Video Module and a 1024x768 LCD panel. The starter board contains an FPGA and several memory devices, including a parallel flash EPROM to store images and a DDR2 RAM memory used to implement a video frame buffer.

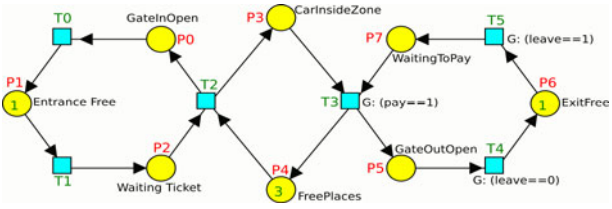

Fig. 3. Demonstration example IOPT model

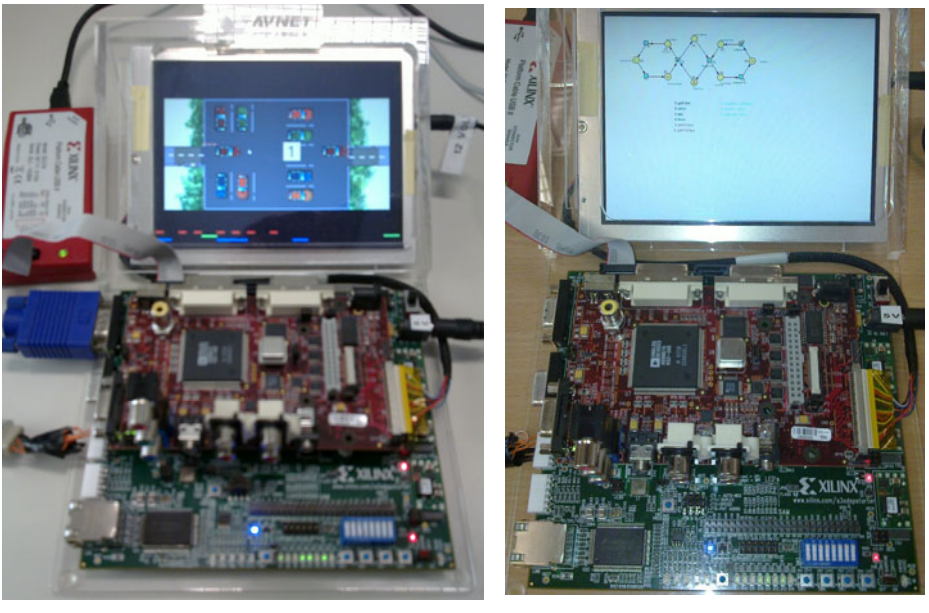

Fig. 4. Prototype photos – Application GUI on the left and debug interface on the right

Figure 4 displays photos of the prototype showing the Parking Lot animation screen and the corresponding debug and diagnostic screen. The car parking lot model contains one entrance, one exit, parking places for up to N vehicles, entry and exit barriers and inputs representing entry and payment sensors.

8 Conclusion and Future Work

An embedded system prototype was created and tested using the proposed tools, demonstrating the capability to rapidly generate embedded applications with sophisticated user interfaces and embedded debug, monitoring and diagnostic interfaces, using IOPT models and without the need to write any line of software code or design hardware components.

The new tools fully automate the task of diagnostics interface creation, generating a solution with minimal hardware requirements, just needing a few hundred Kbytes of EPROM space to store the debug interface images and some FPGA space to implement the animation rules. However, as most rules share code with the controller implementation, the VHDL logic optimizer tools will greatly simplify the generated hardware, removing duplicate signals. For example, both the controller and debug animation modules check if transitions are autonomously enabled and will share the same hardware. Image compression techniques, as simple as RLE encoding, can further reduce the total EPROM memory consumption to less than 50Kb per animation screen, with performance gains and no additional complexity.

Although the prototype was implemented using a standard FPGA development kit, it is possible to use the same tools to create very low cost production embedded controllers using dedicated PCB boards, requiring just one FPGA/ASIC chip, 2Mb of video RAM memory, one EEPROM, voltage regulators, interface logic and connectors, competing with the equivalent micro-controller based solutions.

As a result, in the near future will be possible to add debug and diagnostic interfaces to many embedded devices with no additional labour cost and irrelevant hardware cost increments, turning all the environmental and sustainability gains described in chapter 2 into a reality.

Future work include image the possibility to add pause and step-by-step execution capabilities to the generated interfaces. While the implementation of step-by-step execution mechanisms during simulation and software execution is trivial, it can pose technical challenges for hardware implementations, specially on asynchronous systems. Pausing and step-by-step execution on real hardware devices can create additional difficulties because certain real-time functions cannot be safely interrupted without the risk of causing permanent damages to external hardware and mechanical components. To solve this problem, additional work must be carried on.

Acknowledgment. The third author work is supported by a Portuguese FCT grant ref. SFRH/BD /62171/2009.

References

[1] Moutinho, F., Gomes, L.: From models to controllers integrating graphical animation in FPGA through automatic code generation. In: IEEE International Symposium on Industrial Electronics (ISIE 2009), Seoul Olympic Parktel, Seoul, Korea, July 5-8 (2009)

[2] Gomes, L., Barros, J., Costa, A., Nunes, R.: The Input-Output Place-Transition Petri Net Class and Associated Tools. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN 2007), Vienna, Austria (July 2007)

[3] Gomes, L., Lourenco, J.: Rapid prototyping of graphical user interfaces for Petri-net-based controllers. IEEE Transactions on Industrial Electronics 57, 1806–1813 (2010)

[4] Billington, J., Christensen, S., van Hee, K.M., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M.: The Petri Net Markup Language: Concepts, Technology, and Tools. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 483–505. Springer, Heidelberg (2003)

[5] Coolahan, J., Roussopoulos, N.: Timing requirements for time-driven systems using augmented Petri nets. IEEE Transactions on Software Engineering, 603–616 (September 1983)

[6] Esser, R.: An object oriented Petri net language for embedded system design. In: Proceedings of the 8th International Workshop on Software Technology and Engineering Practice (STEP 1997) (including CASE 1997), p. 216. IEEE Computer Society, Washington, DC (1997)
[7] Chachkov, S., Buchs, D.: From an abstract object-oriented model to a ready-to-use embedded system controller. In: 12th International Workshop on Rapid System Prototyping, Monterey, CA, pp. 142–148 (June 2001)
[8] Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Pratical Use. Basic Concepts, vol. 1. Springer, Berlin (1997)
[9] Kummer, O., Wienberg, F., Duvigneau, M., Cabac, L.: Renew – User Guide. University of Hamburg, Department for Informatics, Theoretical Foundations Group, Release 2.2, August 28 (2009)
[10] Ehrig, H., Ermel, C., Taentzer, G.: Simulation and animation of visual models of embedded systems. In: 7th International Workshop on Embedded Systems Modeling Technology, and Applications, pp. 11–20 (June 2006)
[11] Westergaard, M., Lassen, K.B.: The Britney suite animation tool. In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 431–440. Springer, Heidelberg (2006)
[12] Jorgensen, J.B.: Addressing problem frame concerns via Coloured Petri nets and graphical animation. In: 2006 International Workshop on Advances and Applications of Problem Frames, pp. 49–58 (May 2006)
[13] Chachkov, S., Buchs, D.: From an abstract object-oriented model to a ready-to-use embedded system controller. In: 12th International Workshop on Rapid System Prototyping, Monterey, CA, pp. 142–148 (June 2001)
[14] Nascimento, P., Maciel, P., Lima, M., Santana, R., Filho, A.: A partial reconfigurable architecture for controllers based on Petri nets. In: 17th Symposium on Integrated Circuits and System Design, pp. 16–21 (September 2004)
[15] Costa, A., Gomes, L., Barros., J.P., Oliveira, J., Reis, T.: Petri nets tools framework supporting FPGA-based controller implementations. In: 34th Annual Conference of IEEE Industrial Electronics, IECON 2008, pp. 2477–2482 (2008), doi:10.1109/IECON.2008
[16] Gomes, L., Barros, J.P., Costa, A., Pais, R., Moutinho, F.: Towards usage of formal methods within embedded systems co-design. In: 10th IEEE Conference on Emerging Technologies and Factory Automation, ETFA 2005, September 19-22, vol. 1, pp. 4–284 (2005), doi:10.1109/ETFA.2005.1612535
[17] Reisig, W.: Petri nets: An introduction. Springer, New York (1985)
[18] Tidwell, D.: XSLT. O'Reilly, Sebastopol (2001) ISBN 978-0-596-00053-0
[19] ISO/IEC JTC1/SC7 N3298, ISO/IEC (2005)