XML-based Representation and Monitoring of CAN Devices

Dipl.-Inform. Dieter Bühler[1], Prof. Dr.-Ing. Gerhard Gruhler[2,3]

[1] Wilhelm-Schickard-Institut für Informatik, Symbolisches Rechnen, Universität Tübingen
[2] Institut für angewandte Forschung in der Automatisierung (IFA), Fachhochschule Reutlingen
[3] Steinbeis-Transferzentrum Automatisierung (STA), Reutlingen

Today, the integration of fieldbus devices into the business LAN of an enterprise is usually accomplished by a proprietary OPC solution. This paper describes a more generic and platform independent approach to represent CAN system information and to manage CAN process data. The CANopen Markup Language (CoML), an XML application, was developed in conjunction with several Java CoML tools to achieve this goal. The main benefit of using XML to describe CAN systems and CAN process data lies in the standardized way to represent structured data enriched by meta-data. XML and the corresponding Document Object Model form a basis for rapid development of CoML applications which provide platform independent access, visualization and storage of CANopen setup information and process data. Due to the intrinsic interchangeability of XML information, CoML documents can be processed and evaluated not only by dedicated CoML applications but also by a wide variety of common XML tools like XML editors or XML databases for example. Since XML documents are especially well suited to be deployed to the WWW, this approach also facilitates convenient access to CAN data via the Internet.
This paper gives an introduction to the CANopen Markup Language, the EDS2CoML translator and the CanInvestigator CAN monitoring tool.

1. Introduction

Large enterprises running lots of automation systems at different locations face the problem to somehow handle the huge amount of process, configuration, and documentation data provided or produced by monitoring systems and configuration tools. The data furthermore tend to be heterogeneous especially if different operating systems and hardware platforms are in use. We aim to move towards a generic central information basis from which integrated but dedicated views can be generated in order to allow convenient tuning and monitoring of fieldbus-based automation systems.

In order to achieve this goal we decided to [15] as the ubiquitous data format. XML documents are used to describe CAN device interfaces and CAN system information, to hold CAN process data and to configure monitoring attributes and (remote) access privileges.

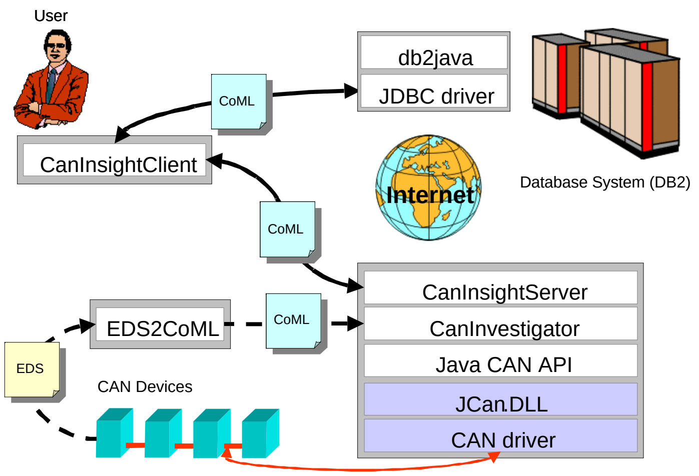

Figure 1: The CANInsight fieldbus management system

XML is an open standard with no operating system or (programming) language dependencies. The *Document Object Model* (DOM) [16] in conjunction with a wide variety of free XML parsers (e.g. [12]) form a convenient basis for rapid XML/ CoML tool development. DOM implementations are available for lots of programming languages like C/C++, Perl and Java. The number of available XML tools (e.g. databases [11], editors [8]) is rapidly growing and all XML applications will automatically benefit from structured data with XML documents results in a maximum degree of manageability and interchangeability of the information within the system and even across the system boundaries.

In this paper we will give an introduction to the *CANopen Markup Language* (CoML), our XML application to represent CAN related data, and present two CoML tools which are actually subsystems of the CANInsight remote CAN management system [4] (cf. Figure 1) which we are developing at the moment. The EDS2CoML subsystem translates CANopen EDS files to valid CoML documents and the CanInvestigator component provides monitoring information of CAN automation systems by creating corresponding CoML documents.

The paper will further present a comparison between the CoML layout and the *Profile Exchange Language* [9] proposed by the IEC.

2. XML and DOM

This section gives a short and informal introduction to XML and the Document Object Model (DOM). Please refer to the XML 1.0 and DOM Level 1 specification for detailed information.

XML documents are plain text files. The character encoding (e.g. Unicode [13]) can be explicitly specified in a declaration part of the document. An XML document is *well-formed* if the document and its sub-entities can be derived from the formal XML grammar given in the XML specification. XML documents are logically structured by tags. A tag consists of a pair of enclosing angle brackets, a tag name and attributes. The part of a document from an opening to the corresponding closing tag is called an XML element. Since in XML the tags have to be balanced (every opening tag must have a corresponding closing tag) and the resulting elements have to be nested correctly within a single root element, the logical structure of every well-formed XML document is a single rooted tree called the *document tree.*

The *Document Object Mode*l defined by the W3C (www.w3c.org) is a set of abstract (programming) interfaces which provide access to the logical tree structure of an XML document. An implementation of these interfaces allows easy creation, modification, and analysis of XML document trees. DOM implementations are available for lots of different programming languages (e.g. *Java API for XML Parsing*

XML is designed as a meta-language and provides the *Document Type Definition* (DTD) concept to specify document classes. Among other things, the DTD lists the known tags, the allowed element nesting and element attributes. If a well-formed XML document meets all restrictions given in a referenced DTD the document is called *valid*. In other words: Valid documents are instances of the document class defined by the corresponding DTD.

3. The CANopen Markup Language (CoML)

The CoML Document Type Definition [5] is a grammar which restricts the set of all well-formed XML documents to a class of documents describing CANopen device profiles, CANopen system configurations and CANopen process data. For each of the named purposes there exists a corresponding XML element which is able to represent the required information.

CANopen Device Profiles: The element *Module* represents CANopen device profiles. The structure of this element and the naming of its subelements and their attributes (cf. Figure 2) are directly derived from the CANopen Electronic Data Sheet (EDS) [6] format. All information that may be contained within an EDS can be represented by a corresponding Module element. In fact the *Module* information normally is a true superset of the EDS information since some additional parameters (e.g. the *Mute* attribute cf. Section 5) were integrated into the device profile description.

The EDS2CoML tool (cf. Section 4) performs an automatic translation from the EDS format to CoML and vice versa. The device profile description can also be written from scratch with a general purpose XML editor (e.g. [8]). The editor can keep track of the correctness of the document by double-checking the referenced CoML DTD while the document is created.

CANopen Process Data: The CanInvestigator component retrieves complete parameter images of (running) CANopen automation systems. The gathered information is represented by a single CoML document within a StateInformation element (cf. Figure 4). The StateInformation element is a compact format for

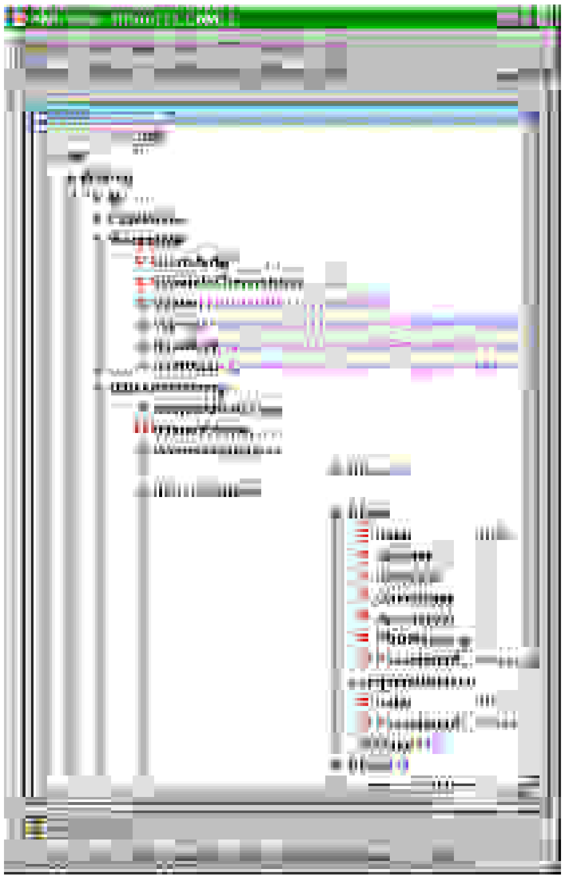

Figure 2: CoML device profile description in a general purpose XML editor

storing current parameter values together with the corresponding CAN module IDs and index values.

A detailed description of CoML is beyond the scope of this paper and can be found in [3].

4. The EDS2CoML Translator

The EDS2CoML tool translates EDS files to CoML documents and vice versa It provides a standard CoML text view and an interactive CoML tree view which provides convenient access to selectable parts of the device profile information. In the EDS text mode the EDS2CoML tool behaves like a standard text editor with cut 'n' paste and search functionality. The translation (in both directions) is initiated just by selecting the corresponding view mode.

The tree view is implemented as a standard Java JTree component. In order to let a JTree work on an XML DOM we created the new class *TreeElement* (cf. www-sr.informatik.uni-tuebingen.de/CanInsight/doc/). This class is

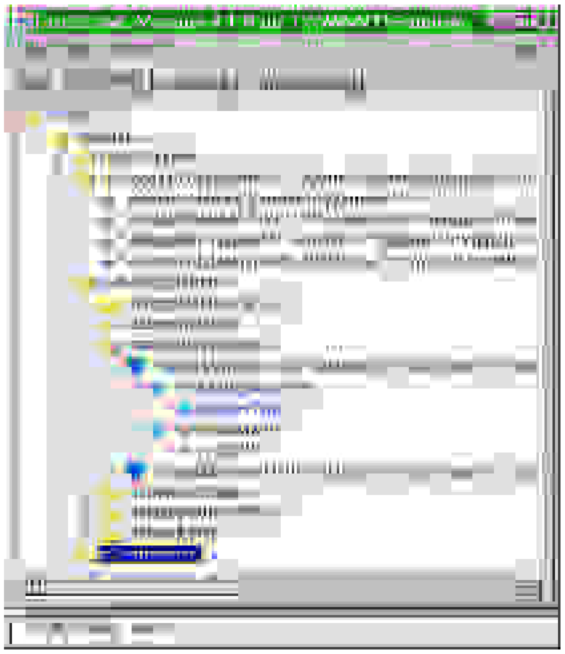

Figure 3: The interactive tree view of the EDS2CoML translator

implements the *javax.swing.tree.TreeNode* interface (which is part of the Java core standard). The implementation of the *TreeNode* interface calls the corresponding DOM methods of the *ElementNode* class to let XML nodes behave like Java JTree nodes.

The allocation of *TreeElement* objects instead of *ElementNode* objects during the DOM creation is configured by passing a corresponding property sheet to the *ElementFactory* object associated with the XML parser instance.

The EDS2CoML translator uses a validating parser and echos all XML errors and warnings to the *Translation Log* view. Thus, all EDS conformance restrictions that are expressible in DTD syntax are automatically checked during the translation and may be used as a first step in an EDS conformance test.

5. The CanInvestigator Monitoring Component

The CanInvestigator CAN monitoring component produces CoML documents containing the parameter images of the connected devices. The monitoring is configured by the *Module* elements (cf. Section 3) of an initialization CoML document which is read by CanInvestigator at startup time. It provides all necessary information to compute an index of all readable parameter values of all connected devices.

mute attribute of the *Object* element, allowing a fine grained definition of the relevant parameter set.

When the monitoring is triggered either by an external request (e.g. from the CANInsight client) or by an internal cyclic signal, the CanInvestigator component requests all indexed parameters via a corresponding CANopen Service Data Object. The retrieved parameter values are decoded according to [7] and stored in a CoML *StateInformation* element which allows a compact representation of the system state (cf. Figure 4). The corresponding *Module* element information is referenced via the *LocationId* attribute of the *StateInformation* element.

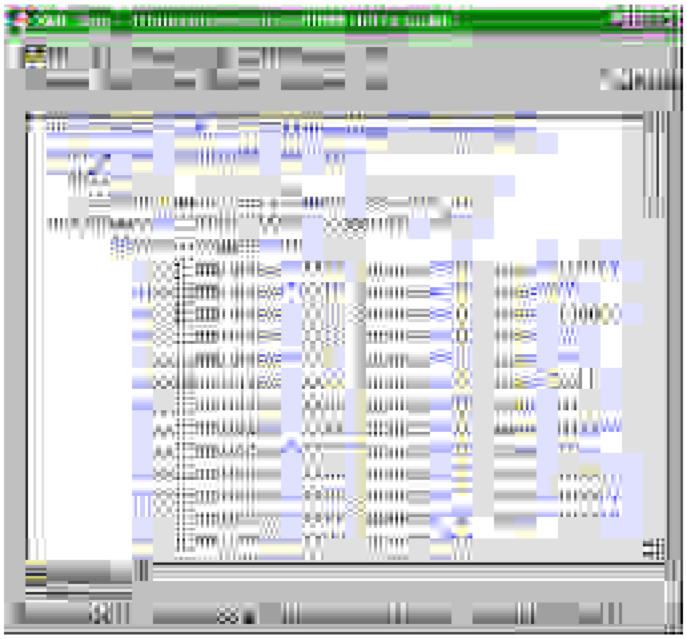
Figure 4: CoML process data

The CanInvestigator is implemented in Java and can be used as a stand-alone CAN monitoring application that produces CoML StateInformation documents with a specific frequency or as a software component providing CoML parameter images from within a different Java application. For example, the CANInsight server uses a CanInvestigator instance to create CoML state information documents which are transferred to remote clients by means of Java Remote Method Invocation.

The access to the CAN hardware is managed by our Java CAN API [2] which forms a convenient framework for rapid CAN/CANopen tool development by encapsulating generic CAN layer 2 messages as well as CANopen

setup we use a CANCardX from Vector Informatik (www.vector-informatik.de) in conjunction with our JCan.DLL [1] which performs the message filtering, message notification and Java to C++ code mediation.

The creation of a complete parameter image of a DIOC711 I/O module from Selectron (www.selectron.ch) for example takes about 770 milliseconds on a Pentium II 350 MHz/WinNT system and a CAN baud rate of 125000.

6. The Profile Exchange Language

The *Profile Exchange Language* (proposed in IEC 61915 [9]) defines a common representation of networked industrial devices and provides a template for documenting that representation independent of the controller device interface used. An XML DTD is provided for the description of generic device profiles.

While the general idea and the specification itself seems to be sound, the provided mapping to an XML representation does not exploit the structural expressiveness of XML. For example, [9, Section 4.5.2.4] describes how to encode the profile classification into one single string value which is mapped onto one XML element with no internal structure. Since the profile classification consists of information about the provided automation function and the device type identifier, it should have a corresponding internal structure that allows direct access to the internal information and should not require further non-XML parsing

There are a few more sections in the specification where the XML specification seems to be too flat in the same way as described above. This seems not to be too hard to handle but there is a further major drawback in the provided XML mapping. As depicted in Figure 5, the value of a described parameter (Par.-Value) is mapped onto the same hierarchical level as the parameter description (Parameter) and not onto a child element of the description like it is done in CoML. This requires the two elements to be linked together via some kind of explicit reference.

Besides the risk of producing dangling references when deleting or updating parameter

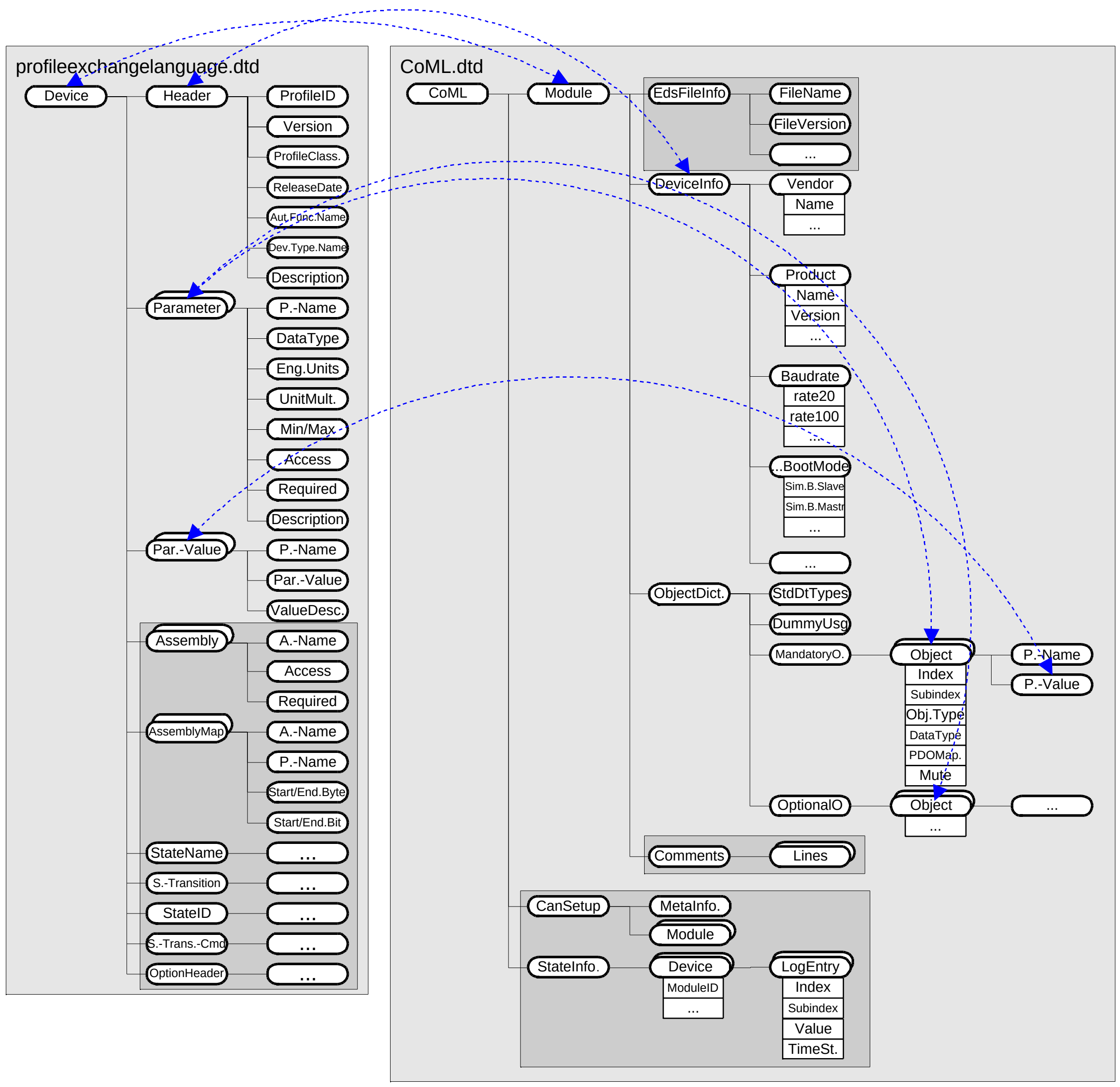

Figure 5: The Profile Exchange Language vs. CoML

pensive to handle by an application. For instance, let the application be interested in the current value of a parameter which is identified by its description. First, the application has to traverse the DOM to retrieve the parameter description and the reference to the parameter value element. In the Profile Exchange Language this reference is the name of the parameter. Second, the application now has to traverse the DOM a second time, until it retrieves the specific parameter value element which holds the same parameter name subelement like the parameter description element does. In contrast, the CoML DOM allows direct access to parameter values via the parameter description since the parameter value element is simply a subelement of the parameter description (cf. Figure 5).

The Profile Exchange Language often uses explicit references in the way described above as would be adequate in the context of relational databases, but in XML this approach makes the resulting document object models hard and expensive to handle. Furthermore, modern native XML databases (like the Tamino system) can be used to store the profile descriptions without the need to break down the hierarchical structure of XML to a flat relational data layout.

In general, it would be possible to extend the Profile Exchange Language with the CoML elements (cf. Figure 5) but at the current stage we would suggest not to do this because of the drawbacks stated above.

7. Related Work

The OPC Foundation (www.opcfoundation.-org) announced in December 1999 that it plans to publish XML schema for OLE for Process Control (OPC) to improve business-to-business and business-to-consumer computing [10]. OPC enables applications to retrieve process data using standardized Component Object Model (COM) interfaces. OPC is inherently restricted to the Windows operating system and is usually addressed by Microsoft Visual Basic or ActiveX applications.

Wollschlaeger [14] discusses the general advantages of describing fieldbus devices with general modeling languages and presents some DTD declarations to describe CANopen device profiles. Issues like monitor configuration or process data representation are not discussed.

8. Summary

XML is a standardized means to represent structured data in a platform and (programming) language independent way. The software industry offers a rapidly growing number of XML tools and programming libraries for all kinds of appliances resulting in an improved manageability and interchangeability of the data.

The CANopen Markup Language provides a means for representing CAN related data in XML. CoML documents can be processed, analyzed, and visualized with dedicated CoML tools or general purpose XML tools. CoML documents are used to represent CAN device profiles, system setup information, process data, and access context related data for system monitoring and remote access facilities.

The EDS2CoML translator can be used to automatically generate CoML device profile representations from EDS information or to generate EDS files from CoML information.

The CanInvestigator component autonomously creates customizable parameter images of (running) CANopen systems by creating corresponding CoML documents. These documents can be stored in XML databases or transferred via a network for remote maintenance and management purposes.

References

[1] D. Bühler, G. Nusser, G. Gruhler, W. Küchlin: „A Java Client/Server System for Accessing Arbitrary CANopen Fieldbus Devices via the Internet", South African Computer Journal, No. 24, Nov. 1999, p. 239 - 243, ISSN: 1015-7999.

[2] D. Bühler, G. Nusser: „The Java CAN API - A Java Gateway to Fieldbus Communication", Proceedings of the 3rd IEEE Workshop on Factory Communication Systems (WFCS 2000), Sep. 2000, Porto, Portugal, IEEE Computer Society Press (to appear).

[3] D. Bühler: "The CANopen Markup Language - Representing Fieldbus Data with XML", Proceedings of the IEEE International Conference on Industrial Electronics, Control and Instrumentation (IECON 2000), Oct. 2000, Nagoya, Japan, IEEE Computer Society Press (to appear).

[4] D. Bühler, W. Küchlin: "Remote Fieldbus System Management with Java and XML", Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE 2000), Dec. 2000, Puebla, Mexico, IEEE Computer Society Press (to appear).

[5] D. Bühler: "The CoML DTD", http://www-sr.informatik.uni-tuebingen.de/CanInsight/CoML.dtd

[6] CAN in Automation e.V.: "Electronic Data Sheet Specification for CANopen", CiA Work Draft 306, Revision 0.3, Sep. 1999, Erlangen, Germany
http://www.can-cia.de

[7] CAN in Automation e.V.: "CMS Data Types and Encoding Rules", CiA DS202-3, Feb. 1996, Erlangen, Germany
http://www.can-cia.de

[8] Icon Informations Systeme GmbH, XML Spy,
http://www.xmlspy.com

[9] International Electrotechnical Commission (IEC): “Low-voltage switchgear and controlgear - Profiles for networked industrial devices”, Draft IEC 61915, Feb. 2000, http://www.iec.ch

[10] OPC Foundation: “OPC and Microsoft start XML initiative”, OPC Quarterly, 2(4), Dec. 1999 http://www.opcfoundation.org

[11] Software AG, Tamino – Native XML Database System, http://www.softwareag.com/tamino/

[12] SUN Microsystems: Java API for XML parsing (JAXP), http://java.sun.com/xml/-download.html

[13] The Unicode Consortium: “The Unicode Standard, Version 2.0”, Addison Wesley Developer Press, Reading Massachusetts, 1996

[14] M. Wollschlaeger: “CANopen Device Descriptions using general purpose modeling languages”, Proceedings of the 6th International CAN Conference (ICC), 1999, CAN in Automation, Erlangen, Germany

[15] World Wide Web Consortium (W3C): “Extensible Markup Language (XML) 1.0”, http://www.w3c.org/TR/REC-xml

[16] World Wide Web Consortium (W3C): “Document Object Model (DOM) Level 1 Specification”, http://www.w3c.org/TR/REC-DOM-Level-1

Acknowledgement
This paper is partially based upon work conducted within the research consortium VVL funded by the German state of Baden-Württemberg through the research initiative *Virtuelle Hochschule*.

[1]Universität Tübingen
WSI – Symbolisches Rechnen
Sand 13
72076 Tübingen, Germany
buehler@informatik.uni-tuebingen.de
www.informatik.uni-tuebingen.de/~buehler

[2, 3] Fachhochschule/STA Reutlingen
Alteburgstr. 150
72762 Reutlingen, Germany
Phone: 0049 7121 271331
Fax: 0049 7121 25713
gerhard.gruhler@fh-reutlingen.de
http://www.fh-reutlingen.de/~www-sta/