A New Method and Format for Describing CANopen System Topology

Matti Helminen, Jaakko Salonen, Heikki Saha*, Ossi Nykänen, Kari T. Koskinen, Pekka Ranta, Seppo Pohjolainen, Tampere University of Technology, *Sandvik Mining and Construction

In this article, we present a new method for describing CANopen network topology. A new format using GraphML, an XML-based graph format, is introduced. By using a subset of GraphML along with CANopen-specific new elements and attributes, topology of single as well as multiple CANopen networks can be captured in a well established graph format with existing tool support. The new format is specified in a manner that allows CANopen design applications to adopt it while providing a mechanism for fallback in unsupported software. Methods for extending the format to contain other CAN- and CANopen-specific data as well as transforming the GraphML-based network structure to other formats are described. Finally, the implications of the introduced method and format are discussed.

1. Introduction and Background

When designing mobile machines in practice, multiple CANopen networks may need to be used. What may then become problematic is that the current CANopen design programs and specification support only description of individual CANopen networks. Specificially, formats and applications for specifying topology between and within networks.

From stricly practical point of view, this may not be problematic. If one wishes, however, to re-use information from CANopen designs, becomes integration of topology information with other CANopen data cumbersome: we not only need to know in which format the topology information is encoded in, but also need to create adapters that (programmatically) combine topology information with other CANopen data. In addition, well-defined points for schema extensions are missing.

Our motivation for enriching CANopen designs with network topology information stems from Semogen project. In Semogen research project, we have attempt to (semi-)automatically generate virtual prototypes, i.e. virtual machine laboratories (VMLs), directly from design data [11, 12, 13, 14]. Within a VML, it is useful to represent CANopen network in visual, graph-like format, where nodes are represented as graph nodes and connection between the nodes as edges in a graph. However since the specific structure within and between nodes is not defined in actual design data, some structure needs to be generated for the purposes of visualization. This is problematic since the visualized structure may or may not correspond to the actual structure of the network. For accurately representing CANopen network structures, we would require to have network topologies defined in design materials, in some common format.

In a more broader level, an important use case for network topology information re-use is CANopen network monitoring and troubleshooting. By enriching for instance onboard device monitors with this structural information, more informative system monitoring tools can be implemented.

From the current formats, the most promising candidate for a point of enrichment is *nodelist.cpj*. As defined by CiA-306-3 [7], *nodelist.cpj* has the main purpose of providing either network or system level structural information and link to appropriate DCF- [16] or XDC-files[17]. What is limiting is that the current format only allows description of individual CANopen networks. DCF-files can be linked to nodes and further to the network, but only locally within a given network. The format also doesn't enable us to describe

in which order nodes are physically connected.
A current effort towards enriching CANopen designs with topology information exist. CiA-302-7 Draft specifies signal routing in gateway nodes, enabling us to track signals between multiple CANopen networks [10]. However, even with CiA-302-7 we still do not have information about topology of the underlying CANopen networks; the way in which the nodes are connected within specific CANopen networks still remain unspecified.

2. Nodelist representation with GraphML

In this chapter we introduce GraphML and define a GraphML-based format for representing nodelists with topology information.This Nodelist.graphml format[9] has also been proposed to CiA.The proposal will add to the future CiA-311 system structure information, which has been integral part of the CiA-306 defining the old design files.

2.1. GraphML

GraphML (Graph Markup Language) is an XML (eXtensible Markup Language) format for graph structures and has a mechanism that allows to define extension modules for additional data [2]. GraphML is well suited for data interchange, because its extension modules allow application specific-data to be added that can be combined or stripped without affecting the graph structure. [1, p.1] This way additional data can be ignored by programs not supporting it without affecting the graph data itself.
GraphML was designed with simplicity, generality, extensibility and robustness in mind. Because of this, the format is easy to parse and interpret and has no limitations with respect to the graph model. GraphML-based formats can be extended with well-defined way to represent additional data and application not capable of supporting this added data can ignore it or extract the subset they can handle [1, p.3].
Format in GraphML is based on XML [2] so there are readily available parsers and tools for it. Also XML enables us to use additional features like Namespaces [3] or XLink [4] inside a GraphML document. In this way, a format can be extended with features required by custom applications.
Structural layer of GraphML describes the fundamental graph model, which is mixed multigraph and may, but is not required to, include nodes, ports, edges, hyperedges and nested graphs [1, p-4]. In figure 2.1.1. we have simple GraphML example file. It has one graph, but GraphML-format may also include multiple graph elements.
Graph-element has mandatory attribute edgedefault, which specifies whether edges are directed or undirected by default. Graphs may contain any number of nodes, edges and hyperedges in any order. In the example we have four nodes and three edges, but no ports or hyperedges.

```
<?xml version="1.0" encoding="UTF-8"?>

<graphml
xmlns="http://graphml.graphdrawing.org/xmlns"
  xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
"http://graphml.graphdrawing.org/xmlns
http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">

  <!-- System description graph (single network) -->
  <graph id="RootNetwork" edgedefault="undirected">
    <node id="Device_A"/>
    <node id="Device_B"/>
    <node id="Device_C"/>
    <node id="Device_D"/>

    <!-- Network connections -->
    <edge source="Device_D" target="Device_A"/>
    <edge source="Device_D" target="Device_B"/>
    <edge source="Device_D" target="Device_C"/>
  </graph>
</graphml>
```

Figure 2.1.1. Simple GraphML example file, which includes graph, nodes and edges.

Port is part of a node to which edges may attach. Ports appear as nested subelements of node. Ports may also include other ports.
Edges connect ports or nodes together. Edge has source and target node, regardless of whether it is directed or not. Edges may also have sourceport and targetport if it is used to connect ports. Edges optional attribute directed can overwrite graphs edgedefault value for current edge.
Hyperedge is a special case of an edge

which can have multiple endpoints. Each endpoints' direction can be defined as in, out or neither. Endpoints refer to nodes and may also refer to ports.

2.2 GraphML extensions

Extensions provide a method for defining additional data and can also provide typing information for it. This additional data in GraphML is defined with the help of key and data elements [1, p8]. A key element is used for typing and naming the data and the data element is for containing actual data value. Every data element has to be linked to ca orresponding key element with id attributes in key and data elements. Multiple data elements should refer to a common key element's id, if they all contain same type of data.
So data typing is done in key element. These elements can be defined in the beginning of a GraphML file. In figure 2.3.2 we see an example of key element definitions in use. Key element has id attribute, which has to be unique and is used to link data element to this key element. Further, for attribute defines what elements (graph, node, edge, port, etc.) can contain this type of data. Purpose of the attr.name attribute is to identify the meaning of the key attribute and it has to be unique, but it is not used inside the document to refer to key attribute [8]. The attr.type attribute defines the type of the data and can be either boolean, int, long, float, double or string. These types are defined like corresponding types in the Java programming language [8].
Data values are defined in a data element, which is located inside graph, node, edge or some other element defined in a key element's for attribute. In our example a data element is located inside graph element as seen in figure 2.3.3. A data element's key attribute refers to key element's id attribute, so we can have typing of our information. The actual value for the data is defined as content of the element. In this case we have property background, which has value of lin_back.bmp as illustrated by figure 2.3.3. Additionally we can see that in corresponding key element (in figure 2.3.2) the background property can be defined for graphs and has to be in string format.

2.3. Nodelist.graphml format

The existing format, nodelist.cpj (figure 2.3.1), represents only nodes within one CANopen network and contains no topology information. The format only allows listing the existing nodes, their DCF-files [16] and names. Yet, having topology information and an overview to the whole CANopen system would be useful for maintenance and troubleshooting and can be used for creating different views of the network or create semantic models of it. Therefore we specified new format, Nodelist.graphml [9], which has also been proposed to CiA as a draft.

```
[Topology]
EDSBaseName=EDS #optional
NetName=DefaultNet #optional
Nodes=0x02
Node2Present=0x01
Node2DCFName=demo_plc.dcf
Node2Name=DemoNode_A #optional
Node3Present=0x01
Node3DCFName=demodeva.dcf
Node3Name=DemoNode_B #optional
```

Figure 2.3.1. Two node network description in nodelist.cpj format.

GraphML provided existing solution for describing graphs in XML based format. It was designed as a data interchange format for graphs and associated data. It also allows defining of extension modules for additional data, in this case node and network information. This additional data does not affect the basic structure of GraphML and information can be added in well-defined way. Even with additional information, GraphML format is still readable and editable in programs that support it. They can present the graph information and leave additional data unchanged. Additional information is decribed with <*key*> and <*data*> tags.
We chose one graph to represent whole CANopen system with multiple CANopen networks. Another solution was considered, where one graph would represent only one CANopen network. In this solution a node would present one CAN node and therefore if one physical

CAN device is connected in multiple CAN networks, it would have had separate nodes in every network's graph.This solution was abandoned due to complexity and the lack of support for multigraphs in GraphML tools. In one graph per whole system solution, one node would also represent one physical CANopen device.This creates the need for having to redefine per network attributes in associated each nodes.
Because the defined GraphML format presents a whole CANopen system and all networks in it, we can attach common attributes to it. To this date, we have defined two attributes [9] at the beginning of a Nodelist file (see figure 2.3.2) These attributes are attached to *graph* element which presents the whole system (see figure 2.3.3.). *EDSBaseName* decribes the path to directory which contains *EDS* files[16]. *Background* attribute is optional and can be used for background image for visualization purposes (for details, see section 3.2 and figure 3.2.2).

```
<!-- Attribute definitions for element <graph> --><key
id="EDSBaseName" for="graph"
attr.name="EDSBaseName" attr.type="string" />
<key id="Background" for="graph"
attr.name="Background" attr.type="string" />
```

Figure 2.3.2. Global parameters defined to graphs.

```
<graph id="RootNetwork" edgedefault="undirected">
<data key="EdsBaseName">EDS</data>
  <data key="Background">lin_back.bmp</data>
  <!-- nodes and edges are defined here -->
</graph>
```

Figure 2.3.3. Setting values to global parameters in graph-element.

```
<node id="N003"> <data
key="NodeName">DemoNode_C</data>
  <data key="NodeType">device</data>
  <data key="NodeFig">testnode.bmp</data>
  <data key="X">220</data>
  <data key="Y">20</data>
  <data key="NumOfNets">1</data>
  <!-- ... -->
</node>
```

Figure 2.3.4. Device specific parameters defined to nodes.

```
<edge id="E002" source="N001" target="N003">
  <data key="NetNumber">7</data>
  <data key="CableName">W5002</data>
  <data key="Length">500</data>
  <data key="LineType">line</data>
  <data key="LineParams"></data>
</edge>
```

Figure 2.3.5. Connection specific parameters defined to edges.

```
<node id="N003">

  <!-- ... -->
  <data key="NetNumberN1">1</data>
  <data key="NetworkNameN1">DefaultNet</data>
  <data key="NodeIDN1">4</data>
  <data key="NodeDCFNameN1">demodevb.dcf</data>
  <data key="SupplyDomainN1">Primary</data>
  <data key="SupplyPointN1">0</data>

  <data key="NetNumberN2">7</data>
  <data key="NetworkNameN2">AdditionalNet</data>
  <data key="NodeIDN2">6</data>
  <data
key="NodeDCFNameN2">demodevb2.dcf</data>
  <data key="SupplyDomainN2">Primary</data>
  <data key="SupplyPointN2">0</data>
</node>
```

Figure 2.3.6. Network specific parameters defined to nodes.

2.4 Discussions

Hyperedges were considered as a way to describe connections between different CANopen networks. The idea was that one hyperedge would join same physical CAN device in different CANopen networks. But the lack of support for hyperedges in GraphML editing tools and added complexity got the idea abandoned. Instead, multiple key values like *NetNumberNx* were defined for nodes with multiple network connections (as in figure 2.3.6) where x describes network number.
Another alternative solution was to use ports as per network connection in nodes. Ports are defined inside the node. This idea has few advantages over using *NetNumberNx* style definition. Firstly, network-specific parameters can be defined in port so there is no need for multiple definitions of a same parameter. For nodes that have multiple different network connections, multiple ports can be used. One for each different network.
This way a schema would be constant for all *nodelist.grapml* files with any number of networks. Also for example *NodeID* is always found in *NodeID* named parameter and not *NodeIDNx* parameter where *x* can be anything. Validation of this type of

format is much easier and filtering or visualizating the network according to parameter specific properties is also much less complicated. Two network example of using port for per-network parameters is presented in figure 2.4.1.
This idea although it was simpler as a format was abandoned due to the lack of port support in GraphML tools and libraries.

```
<!-- ... -->

<port id="P01">
  <data key="NetNumber">1</data>
  <data key="NetworkName">DefaultNet</data>
  <data key="NodeID">4</data>
  <data key="NodeDCFName">demodevb.dcf</data>
  <data key="SupplyDomain">Primary</data>
  <data key="SupplyPoint">0</data>
</port>
<port id="P02">
  <data key="NetNumber">7</data>
  <data key="NetworkName">AdditionalNet</data>
  <data key="NodeID">6</data>
  <data key="NodeDCFName">demodevb2.dcf</data>
  <data key="SupplyDomain">Primary</data>
  <data key="SupplyPoint">0</data>
</port>
<!-- ... -->
```

Figure 2.4.1. Network specific parameters defined to ports.

3. Application Examples

In this chapter we present examples of applying the specified format in two use cases: network visualization in existing applications and a generic system monitoring view implemented in a real control system.

3.1. Network Visualization

Since *Nodelist.graphml* format is GraphML-based, existing GraphML applications and libraries such as *Graphviz*, *yEd*, *Gephi*, *igraph* and *networkx* can be used for purposes of network visualization.
Two examples of network visualization with Gephi are considered: 1) visualization of network topology within a single network, and 2) visualization of multiple networks their connections.
First, let us consider a single star topology network as defined in section 5.3 of the format specification [9]. In this given example, we have a switch - or a hub - acting as a centre point of a physical network, without isolating the branches logically. A DCF file is assigned to the switch, because it is assumed to be a managed switch. This network has been formally defined in GraphML as according to the corresponding listing [9].
Figure 3.1.1. presents visualization of this network in Gephi. Visual parameters are derived as follows: node positions are read from GraphML (*X* and *Y* data fields), Gephi is configured to display node and edge titles (*NodeName* and *CableName* data fields). Nodes are represented using Sphere 3d shape. Note that the visualization demonstrates only one possible way of re-using network data from a GraphML nodelist - other data fields could have been chosen for visual mapping as well.

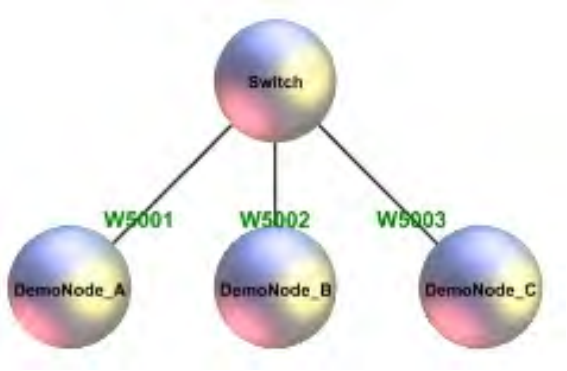

Figure 3.1.1. Visualization of single star-topology network with Gephi.

Secondly, let us consider how a topology of a multi-network CANopen system could be visualized. Consider a minimal example in which two nodes, *DemoNode _B* and *DemoNode _C* have been connected together by a gateway node *DemoNode _A*. Such a network has two separate CANopen networks where the gateway node is a device in the both of these.

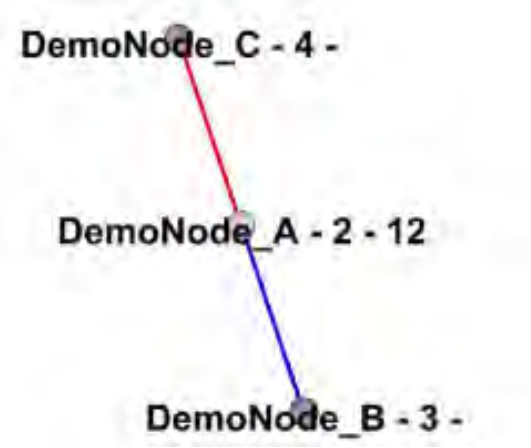

Figure 3.1.2. Multi-network CANopen system visualization with Gephi

3.2. System Monitoring

In this section, we will briefly introduce implementation of a system monitoring view, which is one of the most important parts in each distributed control system GUI, that utitilized the core features of the *Nodelist.graphml* format. The specifics of the implementation are somewhat extensive and outside the scope of this work and thus, are not being introduced.

For actual working machines, it is benefial to provide tools for system monitoring. System monitoring enables us to understand the status of CANopen devices in a system, for instance in attempt to troubleshoot problems within it. In figure 3.2.1, a device monitor view in a system GUI is illustrated. All information used by the view are directly exported from a corresponding CANopen project. Due to limitations of *nodelist.cpj* file, only system name, device names and states can be presented without any relation to the system layout. Cable breaks are among the most common failures, but any connection analysis can not be performed without more specific information on the system's structure.

Figure 3.2.1. Onboard device monitor based on information from nodelist.cpj.

In figure 3.2.2, an alternative system monitor application utilizing data from a *Nodelist.graphml* file is presented. Following the topology information, we can now model actual connections between specific devices in the network. With such enrichment, troubleshooting can be improved by drawing the device status indicators in the correct locations on the system outline. For instance, if based on the network status, we see that specific block of devices are offline, topology information can be used together with the status to deduce locations of problems in physical wiring.

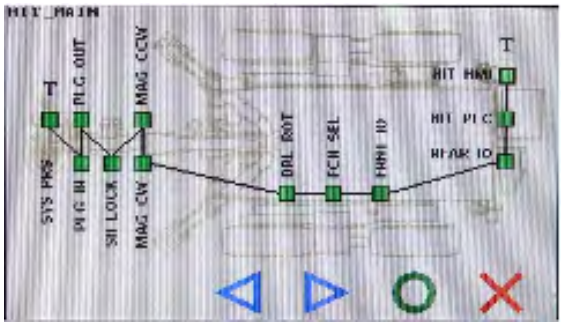

Figure 3.2.2. Onboard system monitor based on information from nodelist.graphml.

In addition to the GraphML-based topology, we also added a 2D visualization of the associated machine structure. By doing so we help the device monitor's user to physically locate devices and their connections and associated problems. Depending on the system complexity and available information, 2.5D background image (similar to what is implemented in SmartSimu Harvester Learning Environment Prototype [18]), could be used as well for potentially more insightful visualization.

3.3. Discussion

In this subsection, we demonstrated how *Nodelist.graphml*, a GraphML-based nodelist format, can be used to visualized structure and parameters of CANopen networks with Gephi visualization tool.
In comparison to data from only *nodelist.cpj*, a GraphML-based format enabled us to create richer visualizations of CANopen networks. Most notably, actual network topology information could be encoded into the visualizations.
The extent in which GraphML features are supported is tool-specific. If a given feature is missing in an application we are required to use, XML transformations such as XSLT could be used for format conversions. By extending this approach to non-GraphML output formats, support for some other, arbitrary visualization tools

could be added. Specifically, with a GraphML to SVG (Scalable Vector Graphics) conversion, support for web browser-based *Nodelist.graphml* viewing could be added.
Further and as presented, we have successfully applied *Nodelist.graphml* data for enrichment of a system monitoring tool. With the help of the topology information, more sophisticated system monitoring view could be implemented. A requirement for the presented, enriched device monitor is that connections between nodes as well as their coordinates in relation to the machine, have been defined in the GraphML data. Ideas for further improving the monitor include representing midpoints in edges illustrating cable routings within the machine. Additionally, properties of power supply could be added to the GraphML data for enabling even more fine-grained troubleshooting capatibilies.

4. Conclusions

In this article, we presented a new method for describing CANopen network topology. A new format using GraphML, an XML-based graph format, was introduced. By using a subset of GraphML along with CANopen-specific new elements and attributes, topology of a single as well as multiple CANopen networks could be captured in a well established graph format with existing tool support. The new format was specified in a manner that allows CANopen design applications to adopt it while providing a mechanism for fallback in unsupported software. Methods for extending the format to contain other CANopen-specific data as well as transforming the GraphML-based network structure to other formats were also described. Two specific usage examples, visualization and system monitoring tool implementation, were also described and discussed.
In terms of applicability, the specified *Nodelist.graphml* format is promising. We have demostrated that the format and associated usage method are applicable to actual design data and informatin re-use use cases. Yet, some further work needs to be done in terms of finalizing the GraphML-based nodelist format, as well as further demostrating its applicability in practical use cases.

References

[1] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M.S. Marshall. GraphML Progress Report: Structural Layer Proposal. Proc. 9th Intl. Symp. Graph Drawing (GD '01), LNCS 2265, pp. 501-512. http://www.inf.uni-konstanz.de/algo/publications/behhm-gprsl-01.ps.gz
[2] W3C. Extensible Markup Language (XML). http://www.w3.org/XML
[3] W3C. Namespaces in XML 1.0 (Third Edition). http://www.w3.org/TR/REC-xml-names/
[4] W3C. XML Linking Language (XLink) Version 1.0. http://www.w3.org/TR/xlink/
[5] Gephi. The open graph viz platform. http://gephi.org
[6] CiA 306 V1.3.0: CANopen electronic data sheet specification
[7] CiA-306-3 Electronic Device Description, Part 3: Network variable handling and tool integration
[8] GraphML Primer http://graphml.graphdrawing.org/primer/graphml-primer.html
[9] Saha, H., Nykänen, O., Helminen, M., Salonen, J. (Eds.). (2011). Nodelist.graphml Format Specification, December 13, 2011. http://wiki.tut.fi/SmartSimulators/NodelistGraphML
[10] CiA 302 Draft Standard Proposal, Part 7: Multi-level networking. Version: 1.0.0, 02 February 2009.
[11] TUT / SmartSimulators. Semogen. https://wiki.tut.fi/SmartSimulators/Semogen
[12] Nykänen, O., Salonen, J., Markkula, M., Ranta, P., Rokala, M., Helminen, M., Alarotu, V., Nurmi, J., Palonen, T., Koskinen, K., Pohjolainen, S.: What Do Information Reuse and Automated Processing Require in Engineering Design? Semantic Process. Journal of Industrial Engineering and Management (2011) (Accepted)
[13] Salonen, J., Nykänen, O., Ranta, P., Nurmi, J., Helminen, M., Rokala, M., Palonen, T., Alarotu, V., Koskinen, K., Pohjolainen S. (2011). An Implementation of a Semantic, Web-Based Virtual

Machine Laboratory Prototyping Environment. In: The Semantic Web – ISWC 2011, Lecture Notes in Computer Science, 2011, Vol. 7032/2011, pp. 221-236.
[14] Markkula, M., Rokala, M., Palonen,T., Alarotu V., Helminen, M., Koskinen, K. T., Ranta, P., Nykänen, O., Salonen, J. (2011). Utilization of the Hydraulic Engineering Design Information for Semi-Automatic Simulation Model Generation. Proceedings of the Twelfth Scandinavian International Conference on Fluid Power, vol. 3, pp. 443-457. May 18-20, 2011, Tampere, Finland.
[15] ISO 11898-2:2003 - Road vehicles -- Controller area network (CAN) -- Part 2: High-speed medium access unit
[16] CiA-306-1 Electronic Device Description, Part 1: Electronic Data Sheet and Device Configuration File
[17] CiA-311 CANopen Device Description, XML schema definition
[18] SmartSimu Harvester Learning Environment Prototype. https://wiki.tut.fi/SmartSimulators/SmartSimuHarvester

Matti Helminen
Tampere University of Technology
PL 527, 33101 Tampere, Finland
phone: +358 3 3115 11
email: matti.helminen@tut.fi
www.tut.fi

Jaakko Salonen
Tampere University of Technology
PL 527, 33101 Tampere, Finland
phone: +358 3 3115 11
email: jaakko.salonen@tut.fi
www.tut.fi

Heikki Saha
Sandvik Mining and Construction
PL 100, 33311 Tampere, Finland
phone: +358 (0) 205-44 121
email: heikki.saha@sandvik.com
www.sandvik.com

Ossi Nykänen
Tampere University of Technology
PL 527, 33101 Tampere, Finland
phone: +358 3 3115 11
email: ossi.nykanen@tut.fi
www.tut.fi

Kari T. Koskinen
Tampere University of Technology
PL 527, 33101 Tampere, Finland
phone: +358 3 3115 11
email: kari.t.koskinen@tut.fi
www.tut.fi

Pekka Ranta
Tampere University of Technology
PL 527, 33101 Tampere, Finland
phone: +358 3 3115 11
email: pekka.a.ranta@tut.fi
www.tut.fi

Seppo Pohjolainen
Tampere University of Technology
PL 527, 33101 Tampere, Finland
phone: +358 3 3115 11
email: seppo.pohjolainen@tut.fi
www.tut.fi