A Formal, Product Structure Driven Design of Optimized End-to-End Demand Supply Chains

Teemu TYNJÄLÄ
Nokia Research Center, Computing Architectures Lab
FIN-00045 Nokia Group, Finland

ABSTRACT

Demand supply planning is a challenging task in today's business environment. Several alternative supply chains and evolving product structures present the planners with thousands of network options. Analysis of possible networks one by one via spreadsheet programs is too time consuming. In this paper we propose a Petri Net based method and algorithms that automatically calculate all demand supply network options, with associated costs, for arbitrary user-defined product structures. The effectiveness of the method inside Nokia is also discussed.

Keywords: Demand Supply Network Modelling, Petri Nets, Reachability Analysis

1. INTRODUCTION

Today's logistics professional faces an increasing challenge from constantly changing product structures, and numerous globally distributed suppliers. Even a simple device may have hundreds of possible supply network options, and a manual, spreadsheet-aided analysis becomes too slow. Normally, the companies have solved the problem by focusing on certain hand-picked suppliers, but in so doing, they may let a cheaper option pass by.

Optimization methods have been used for over 50 years to solve similar problems, but they are mostly designed to find the optimum quickly, and leave the rest of the options unexplored. In today's business environment, risk analysis must also be performed [4], and a logistics professional is interested in knowing all possible supply networks, irrespective of the cost.

The research was started with the following question: "What is a suitable formalism that supports automated analysis (complete state space) and allows the integration of product structures with their demand supply networks?" In the course of research, we developed a Petri Net based methodology, and algorithm that extends the traditional reachability analysis. The method has been implemented inside Nokia corporation, where it is in daily production use.

This paper presents the developed *DSNnet* formalism, and discusses the method's effectiveness inside Nokia. Section 2 presents the network formalism. Section 3 presents pseudocode for the reachability analysis algorithm. Section 4 describes the Nokia implementation of the *DSNnet* method. Section 5 gives user experiences and performance data of *DSNnet* tool in Nokia. Section 6 presents related works, and Section 7 concludes.

2. DSNnet FORMALISM

The *DSNnet* formalism is based on extended Petri Nets, where AND and OR nodes are included in the network description. This enables the construction of nets where e.g. only one arc is chosen after a transition fires (XOR logic). The extended Petri Nets were introduced and developed by Agerwala [1] and Baer [3], who also proved that such extension to the Petri Net language does not increase their expressive power. The following presentation of *DSNnet* formalism does not give an exact mapping to the underlying Petri Net structure, but such developments may be found [6].

The first subsection introduces the net formalism, and the second describes a case of deriving a *DSNnet* from a simple product structure. This example has been taken from [6].

DSNnet Definition

The formal definition of *DSNnet* is shown in Figure 1. The *DSN_net_skeleton* defines the type of cost elements that may be assigned to nodes and arcs, and a *DSNnet* is a concrete network instance. The list below describes the elements *DSN_net_skeleton*.

- *NodeTypes* describes the types of nodes to be included in the analysis (e.g. demand nodes, manufacturing). Node Types always includes the special AND and OR nodes that enable the modeller to represent aggregation and choice.
- *ArcTypes* describes the types of arcs that may be included in the analysis. Usually these include transport arcs.
- *ParameterPool* is the collection of all parameters that will be taken into account in the computation of Demand Supply Network total cost, e.g. Inventory Carrying Cost, Freight cost.
- *TypingFunction* affixes cost parameters to NodeTypes and ArcTypes.

Typically, a *DSN_net_skeleton* is company specific. The cost elements included in the analysis, and parameter values for inventory carrying cost differ from one industry to another. The *DSN_net_skeleton* may also evolve during the course of a company's lifetime. In one sense, a *DSN_net_skeleton* is a metamodel or profile for a *DSNnet*.

An actual *DSNnet* instance is based on one *DSN_net_skeleton*. This instance is Petri net-like, with nodes, arcs and a flow relation. The only difference to a typical Petri net is the

$$
\begin{array}{lcl}
DSNnet & = & \{Nodes, Arcs, F, DSNnet\_skeleton\} \\
DSNnet\_skeleton & = & \{NodeTypes \cup \{AND\_node,\ OR\_node\}, ArcTypes, ParameterPool, TypingFunction\} \\
 & & \\
NodeTypes & = & \text{Types of nodes in the analysis. E.g. demand nodes, manufacturing nodes} \\
ArcTypes & = & \text{Types of arcs in the analysis. E.g. transport arcs, simple connectors} \\
ParameterPool & = & \text{Collection of all parameters used in cost analysis} \\
TypingFunc & = & NodeTypes \times 2^{ParameterPool} \cup ArcTypes \times 2^{ParameterPool} \\
 & & \\
Nodes & = & NodeId \longrightarrow NodeTypes \cup \{AND\_node, OR\_node\} \\
Arcs & = & ArcId \longrightarrow ArcTypes \\
Restriction\ on\ F & = & F \subseteq Nodes \times Arcs \cup Arcs \times Nodes
\end{array}
$$

Figure 1: DSNnet definition

inclusion of AND and OR nodes to describe aggregation and choice, respectively. An example of a *DSNnet* derived from a product structure is discussed next.

DSNnet Example [6]

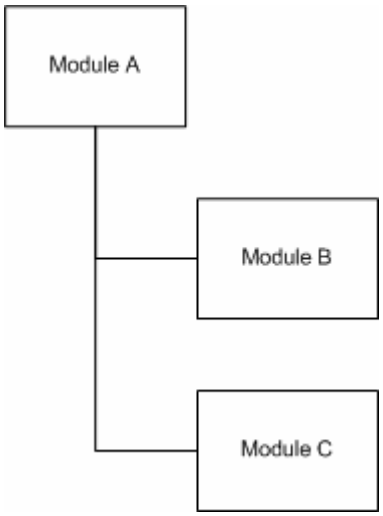

Figure 2: A simple product structure

Consider a simple product structure given in Figure 2. A logistics professional may be faced with the following constraints when deciding on a demand supply network:

- Module A will have only one supplier
- Module B can be sourced from two suppliers, and the sourcing options are:
  - purchase all components from "Supplier 1 for Module B"
  - buy 60% of the volume from "Supplier 1 for Module B", and 40% of the volume from "Supplier 2 for Module B".
- Module C has two suppliers and the sourcing options are:
  - buy 100% of the volume from "Supplier 1 for Module C"
  - buy 100% of the volume from "Supplier 2 for Module C"

Faced with these constraints, the *DSNnet* formalism allows the construction of a network as depicted in Figure 3. Notice that for the ease of presentation we do not include the actual parameter mappings in this example, nor the final customer of Module A.

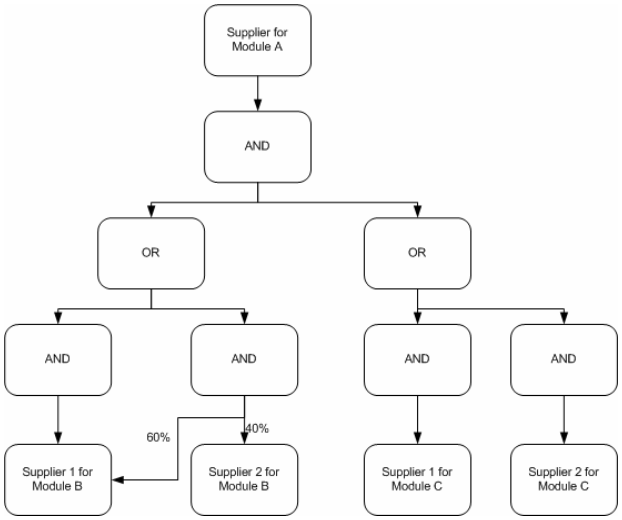

Figure 3: DSNnet for the product in Figure 2

3. ANALYSIS OF DSNnet's

DSNnet formalism may be analyzed via reachability analysis, a central analysis method for Petri Nets. In the present work, the traditional reachability analysis algorithm was modified to allow several initial states in the *DSNnet*.

The algorithm consists of three steps, First, traditional reachability analysis algorithm is used to compute the possible *DSNnet*'s for each customer separately. Next, these reachability graphs are combined to produce full paths for the Demand Supply Network setup. Finally, the volume dependent investment costs for each full path are calculated.

The next subsection presents the pseudocode that implements the analysis algorithm. The cost calculation routine details are omitted as these are *DSN_net_skeleton*-specific. However, we do indicate where they are called during algorithm's execution.

```
Compute_DSN_Paths_For_Multiple_Markets(array demand_nodes)
{
   /* Do reachability analysis  for each demand node  */
  For each demand node do (i = 1..number of demand nodes) {
     DSN_paths[i]:= Compute_DSN_Paths(demand_nodes[i]);
  }

  Total_DSN_Result := empty array;

 /* Combine the results from previous step */
 For all DSN paths do (i := 1..number of demand nodes) {
    Total_DSN_Result := Concat(Total_DSN_Result, DSN_paths[i]);
 }

 /* Add volume dependent investment costs */
 For all paths in Total_DSN_Result do (i := 1.. number of paths) {
   Total_DSN_Result[i] :=AddInvestmentCosts(Total_DSN_Result[i]);
 }

  Return Total_DSN_Result;
}
```

Figure 4: Main Analysis Routine

The Algorithm

The algorithm is divided to three parts: the main routine, the reachability analysis algorithm for a single demand node, and the routine for calculating investment costs. The three parts of the algorithm are shown in, respectively, Figure 4, Figure 5 and Figure 6.

The first helper function, *Concat(array1, array2)*, determines all the path combinations where the first path is taken from array1 and the second from array2. *Concat* adds the costs from the component paths to get the cost of the aggregated path. The result of the function is returned as an array. Cost calculation routines for nodes and arcs will be highly dependent on the cost elements we include in each NodeType and ArcType. The second helper function, *AddRows(array1, array2)*, simply adds all rows from array2 to array1.

Creating a DSNnet from Product Structure

The above algorithm is integrated to a company's product database to enable push-button demand supply network analysis. The following procedure is one way of making the 'conversion' from a product and supplier database to *DSNnet* format.

1. The user must specify the factory/suppliers the parent item will be supplied from
2. The user must select the 'interesting' child items. Only these components will be included in the analysis.
3. The user must specify a sourcing matrix for each of the suppliers supplying the parent item. More precisely this will force the user to decide who will supply the 'interesting' child items in each case.
4. Each of the 'interesting' child items now becomes a parent item and we go back to step 2.

Figure 7 shows what a generalized *DSNnet*, translated via the procedure, will look like.

```
Compute_DSN_Paths (start_node, result_array)
{
   if (start_node has 0 children) {
      add start_node to each path in result_array;
      add start_node's costs to each path in result_array;
      Return result_array;
    }

  else if(start node has 1 child) {
      assign start_node's volume to child arc & node;
      add start_node to each path in result_array;
      result_from_below := Compute_DSN_Paths (child, result_array);
      add start_node's costs to each path in result_from_below;
      add the arc's costs to each path in result_from_below;
     Return result_from_below;
  }

  else if (start_node is ANDnode)
  {
      add start_node to each path in result_array;
      i:=0;
      forall i: intermediate_result[i] := empty array;
      for each child of start_node do (i := 1..number of children) {
        sub_result[i] := Compute_DSN_Paths (child[i], empty array);
      }
      consolidated_result := empty array;
      for each child of start_node do (i := 1..number of children) {
        consolidated_result := Concat(consolidated_result, sub_result[i]);
      }
     AND_Result := Concat(result_array, consolidated_result);
     Return AND_Result;
  }

  else if (start_node is ORnode)
  {
      add start_node to each path in result_array;
      i:=0;
      forall i: intermediate_result[i] := empty array;
      for each child of start_node do (i := 1..number of children) {
        sub_result[i] := Compute_DSN_Paths (child[i], empty array);
      }
      OR_result := empty array;
      for each child of start_node do (i := 1..number of children) {
        OR_result := AddRows(OR_result,
                                  Concat(result_array, sub_result[i]));
      }
     Return OR_Result;
  }
}
```

Figure 5: Reachability Analysis Routine for a Single Demand Node

```
AddInvestmentCosts(path)
{
   hash_table := empty hash table;
   for each node in path do
   {
      if (node exists in hash_table)
         add volumes of the nodes and record result in hash_table;
      else
          insert node into hash_table;
   }

  for each node in hash_table do
  {
     determine volume dependent investment costs;
      update path cost;
  }

  Return path;
}
```

Figure 6: Investment Cost Analysis Routine

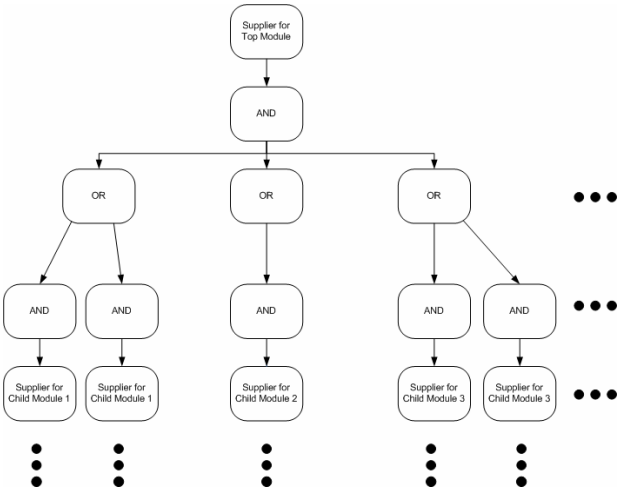

Figure 7: A Generic DSNnet Structure

Notice that there are several AND and OR nodes in the picture. This is necessary to allow the user to specify production splits between several different suppliers.

4. IMPLEMENTATION INSIDE NOKIA

The above methodology was implemented inside Nokia during 2004 as a web-based service. The product information is stored in an Oracle database, and the reachability analysis is implemented in Java. The workflow in Figure 8 illustrates a normal use of the tool.

The user will first create a product structure with supplier alternatives using the web interface. Next, the analysis routine is invoked. The analysis routine is divided into two parts: the Java code first generates a *DSNnet* from product information in the database, and then the reachability analysis routine is called on the generated net. Finally, the results are stored in the database and shown to the user.

The coding of the translation routine from Oracle to Java and vice versa was the most challenging programming effort. It took nearly 5 months to complete, whereas the reachability analysis routine was finished in 2 months.

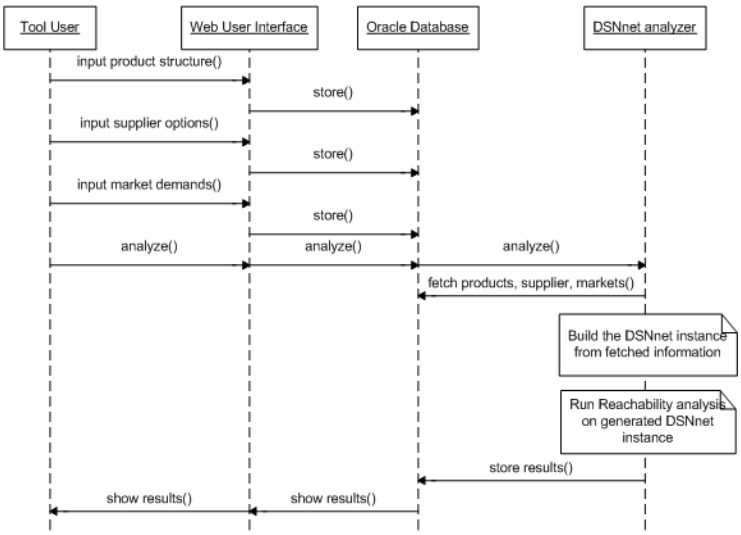

Figure 8: Workflow of DSNnet Analysis Tool inside Nokia

5. USER EXPERIENCES AND EXPERIMENTAL RESULTS [6]

The tool has been received very well inside Nokia. The productivity of the logistics personnel has increased dramatically, and the tool is in active use in every business unit of Nokia. In one year of operation, the logistics managers have analyzed 170 real life cases, reaching from base station distribution to Pakistan, to global supply of N70 smart phone.

The performance of the tool is reasonable as the following chart shows. Before, a logistics professional was able to analyze 5 network setups per day using a spreadsheet program. The tool provides a hundredfold increase in productivity, and assures the good development of logistics practices in Nokia.

The performance is similar in all parts of the world, since the analysis time dominates any transmission delays. Hence, whether a Nokia logistics professional is in San Diego or in Tokyo, the response times are the same.

# of DSN alternatives	Analysis time (seconds)
4	5
16	6
64	30
256	71
1024	330

6. RELATED WORK

Recently, simulation-optimization [2] and genetic algorithms [5] have been proposed as tools to demand supply network analysis. The difference between these methods and *DSNnet* deals with the state spaces. Reachability analysis analyzes the whole state space, which is not the case with optimization algorithms. However, optimization techniques enable the analysis of larger problems if only the optimal solution is needed. In the current climate of supply chain risks, complete state space analysis for demand supply networks seems prudent.

7. CONCLUSION

We developed a Petri Net based formalism with reachability analysis that allows logistics professionals to analyze demand supply networks in a product-driven manner. Reachability analysis guarantees that we search the complete state space of possible demand supply networks, and are quick to react to supply chain risks.

A tool based on the methodology was implemented in Nokia during 2004. The tool is used currently in all business units and the feedback has been very positive. The users have thanked especially the linkage between a product and its demand supply network.

The tool will be further developed inside Nokia. The future developments will improve tool useability, reduce the exponential time complexity of the reachability analysis algorithm, and integrate the tool to an external engine for dynamic simulation of networks.

REFERENCES

[1] Agerwala, T., Flynn, M., Comments on capabilities, limitations and "correctness" of Petri nets, Proceedings of the 1st annual International Symposium on Computer Architectures (ISCA '73), **ACM SIGARCH Computer Architecture News**, Vol. 2, Issue 4, 1973.

[2] April, J., Glover, F., Kelly, J.P., Laguna, M., Practical Introduction to Simulation Optimization, **Proceedings of the 2003 Winter Simulation Conference**, S. Chick, P.J. Sanchez, D. Ferrin, and D.J. Morrice, eds.

[3] Baer, J.L., Modeling for Parallel Computation: A Case Study, **Proceedings of the 1973 Sagamore Computer Conference on Parallel Processing.**

[4] Norrmann, A., Jansson, U., Ericsson's proactive supply chain risk management approach after a serious sub-supplier accident. **International Journal of Physical Distribution & Logistics Management**, Vol. 34, No. 5, 2004.

[5] Truong, T.H., Azadivar, F., Simulation-Based Optimization for Supply Chain Configuration Design, **Proceedings of the 2003 Winter Simulation Conference**, S. Chick, P.J. Sanchez, D. Ferrin, and D.J. Morrice, eds.

[6] Tynjälä, T., Supporting Demand Supply Network Optimization with Petri Nets, Submitted to **International Conference on Application and Theory of Petri Nets (ICATPN) 2006.**