# DESIGN PATTERNS FOR LOOSELY COUPLED TRACK, TRACE, CONFIGURATION, AND CHECK OPERATIONS IN MULTI-COMPANY ENVIRONMENTS

*Jan Holmström and Kary Främling*

*Department of Industrial Management and Engineering*
*Helsinki University of Technology*
*PO Box 5500, FIN-02015 HUT, Finland*

## ABSTRACT

Operations design in a network of suppliers and service providers requires a scheme by which the operations are assigned to the suppliers and service providers. This paper introduces a scheme for pattern-based operations design in an evolving multi-company network (i.e. "virtual enterprise"). It presents operations design patterns for solving four problems of managerial importance for customizing products in multi-company operations environments: (1) track and trace of shipments; (2) track and trace of composite products; (3) dynamic order structures; (4) planning and scheduling of composite products and dynamic order structures.

**Keywords:** Design Patterns, Product Customization, Virtual Enterprise

## INTRODUCTION

The paper is foundational, synthesizing software engineering (Gamma et al., 1995, Främling et al., 2004) and product centric supply chain management (Holmström et al., 2002). The approach is to combine fragments of existing theory and knowledge into a new representation scheme for loosely coupled operations design. This scheme is used to illuminate how loosely coupled operations design facilitates customizing and adaptability in logistics networks. The paper is intended to raise awareness of the far reaching implications of object oriented operations architectures for creating operations designs where products are easy to configure and collaboration with new partners is easy to start up and finish.

The paper consists of four sections. Section 2 introduces two basic representation schemes for operations design: procedural-based workflow and pattern-based object schemes. The advantages in applying the pattern-based object scheme in a changing network environment are outlined based on a literature review. Section 3 develops basic pattern-based object schemes for logistics operations design. The operations designs are: (1) track and trace of shipments; (2) track and trace of composite products; (3) dynamic order structures; (4) planning and scheduling of composite products and dynamic order structures. Finally, section 4 summarizes the key theoretical insights and discusses how to start operations design in practice.

## LITERATURE REVIEW: SCHEMES FOR REPRESENTATION

To design loosely coupled operations an effective scheme is needed that assigns operations to service providers without redesign and excessive setup. There are two basic schemes by which value adding operations can be described and allocated to service providers. These are procedural-based workflow and pattern-based object schemes.

The idea of the two basic schemes is taken from Petrie and Bussler (2003, p. 74) who call the pattern-based schemes condition-based. We could also have used Simon's (1996, p. 210) classification of state descriptions and process descriptions. The problem with using Simon's definitions is that in Business Process Re-engineering what is referred to as a "process" really is a state description, and the pattern-based scheme we are introducing as a new design tool is a process description according to Simon's classification. This is why we do not use this better known classification. It would unnecessarily confuse the point we try to make in this paper.

The **procedural-based workflow** scheme is widely used in logistics operations design today. The scheme gained prominence through business process re-engineering (Hammer, 1990) and redesign (Davenport and Short, 1990). It describes and assigns operations by defining the work-flow or process between participants and resources. In business information systems the process is executed by transactions that transform the product and transfer it between points. Today determined attempts are being made to extend the scope of procedural-based workflow operations beyond the confines of the individual enterprise. Supply Chain Event Management solutions have been developed to monitor the progress of complex operations in the extended supplier network (Otto, 2003).

However, the product is a passive object of control that is scheduled through a sequence of value adding operations as defined by the procedural-based workflow scheme. The passive role of the product individual in the representation scheme makes it difficult to use when designing loosely coupled logistics solutions in changing network environments. The scheme does not support operations design of customized products that can be used in different situations without redesign and with minimal prior arrangement. The root problem is that the procedural-based workflow scheme fixes the order of operations in the representation, and if individual products or customers require a different order of operations this needs to be described as a separate process. The need to describe the process for each constellation of participants and type of order results in a "spaghetti" of descriptions (Holmström et al., 2002).

Using **pattern-based object** schemes helps embedding control and decoupling interaction between the individual objects which improves the adaptability of the design as well as reducing the need for redesign. Pattern-based object schemes have primarily been developed and used in object oriented programming. Proven designs have been documented in the form of design patterns describing the problem situation, solution design and consequences of the design (Gamma et al, 1995). The patterns approach was originally developed in architecture (Alexander, 1977).

In pattern-based object schemes the operation is represented by identifying the participating objects and describing the communication between these objects. A simple example (Gamma et al., 1995, p. 33) is a drawing program where the components of the drawing are represented as interacting objects. Integration is achieved by objects performing operations on other objects, which in turn change the attributes of those objects. When designing the drawing software according to tested design patterns there is no need to anticipate the order of operations exactly, nor the process of how the user will draw his figure, as would be the case with a design based on a workflow scheme.

The goals of the design patterns are typically related to reducing redesign and improving adaptability in changing circumstances. According to Gamma et al. (1995, p. 2) tight coupling is a major source of redesign in object-oriented systems. Classes that are tightly coupled are hard to reuse in isolation, since they depend on each other. Tight coupling leads to monolithic systems, where you can’t change or remove a class without understanding and changing many other classes. The system becomes a dense mass that is hard to learn, port and maintain. Loose coupling increases the probability that a class can be reused by itself and that a system can be learned, ported, modified, and extended more easily.

The object oriented programming concepts *object* and *class* correspond to entities involved in loosely coupled operations design as follows (Främling et al., 2004). An *object* is an *individual* product item, supplier, employee, customer etc. that constitute a part of the economic network. From the point of view of designing loosely coupled operations solutions the most important is the individual product. The *class* corresponds to the *types* of objects in the economic network, i.e. product, supplier, consumer. Individual objects are linked by globally unique object references. At its simplest the object reference is an ID@URI pointer to a product specific entry in a company database.

Examples of using pattern-based object schemes for operations design can be found in literature. A central organizing concept in the examples is the product agent or product holon. A recent example from logistics operations design uses software agents for representing individual products (Kärkkäinen et al., 2003). This way it was possible to implement item tracking and tracing as a product attribute instead of having to link information from different enterprise systems along the delivery path. A similar example is linking an adaptive project management solution to the model of a building under construction. Petrie et al. (1999) refers to this as agent based project management. Examples also exist for controlling material flow inside a factory (Gausemeier and Gehnen, 1998); and for the design of an adaptive video editing process (Davis, 2003).

To sum up, a representation scheme that can be used for designing loosely coupled operations in a changing network environment is needed. Workflow based representation is sufficient for developing operations within the scope of control of a single enterprise, but insufficient for a dynamic network. Integration in a changing environment cannot be accomplished by workflow integration. Instead it requires pattern-based object control schemes (Petrie and Bussler, 2003, p. 75; Holmström et al, 2002?). Table 1 summarizes the difference between loosely coupled operations design and conventional operations management.

*Table 1. Summary of differences between conventional operations management and loosely coupled operations design*

| | **Operations management** | **Operations design** |
|---|---|---|
| **Goal** | Efficient and reliable operations | Loosely coupled operations design |
| **Economic network environment** | Stable products and processes within the scope of control of a single enterprise | Customized products, dynamic virtual enterprise |
| **Representation method** | Procedural-based workflow | Pattern-based objects |

## PRESENTATION OF LOOSELY COUPLED OPERATIONS DESIGNS

This section presents pattern-based object schemes for loosely coupled operations design. The sub-sections articulate how specific design patterns can be used in a network environment to design the adaptive solutions needed for managing information in a virtual enterprise producing, delivering, and servicing customized products. The subsections are on: (1) track and trace of shipments; (2) track and trace of composite products; (3) dynamic order structures; (4) planning and scheduling of composite products and dynamic order structures.

*Track and trace of shipments*

Shipment tracking and tracing is a basic functionality needed for customizing and for implementing the virtual enterprise. If individual products and shipments cannot be identified and located it is impossible to customize items in a workflow.

Tracking and tracing of shipments is currently performed by web portals where a shipment number is used to access the location of a shipment. Companies that need to track a great number of shipments also have the possibility to create a direct connection to tracking information by EDI messages or similar.

This operations design is an example of tight coupling because the way to access information depends on the provider of the information. Connections between companies are often set up as one-to-one links with specific messages and message formats. Adding a connection to a new company may require the development of a new messaging interface and modifying the existing software or creating a new one. The result is that visibility of inventory and control of product individuals is poor in most industries. However, improvement is crucial because the high costs related to product recalls and increasing industry specific requirements from authorities (Töyrylä, 1999). What is needed to resolve the problem is to replace this tight coupling with a loosely coupled design.

Design patterns in object oriented programming use techniques such as abstract coupling and layering to promote loosely coupled systems (Gamma et al., 1995, p.25). The most interesting pattern for designing a loosely coupled tracking and tracing solution is the Observer pattern (Främling et al., 2004). The Observer pattern is recommended in situations when an object should be able to notify other objects about a change in the object, without making assumptions about who these objects are. The number of objects to notify may evolve with time. This is the case of tracking and tracing in a multi-company environment where it is difficult to define beforehand how many participating companies may be involved in tracking and tracing a product, for supply chain visibility and product security reasons. To support tracking and tracing in a changing participant network it is a great advantage if as few as possible assumptions about the participants need to be made when implementing the solution design.

The key objects in the Observer pattern are **Subject** and **Observer,** and the associated **ConcreteSubject** and **ConcreteObserver** objects. The ConcreteSubject maintains a reference list to the Observer objects. In the case of tracking in a logistics network, when the shipment's product agent (the ConcreteSubject) receives a location update message from a tracking checkpoint, it notifies all registered Observer objects about the update.

The first step in order to implement the outlined Observer pattern for tracking and tracing in a changing multi-company network is decoupling the tracking checkpoints and the product agents. This is achieved by attaching the ID@URI reference of the product agent to the product or shipment on an RFID-tag, barcode or some other identifier as described in Kärkkäinen et al. (2002). Therefore it is sufficient to download a generic client software component for creating a new tracking checkpoint. This way all the handling points and the shipment's product agents are linked over the Internet by using a simple ID@URI notation. In the Observer pattern, the shipment's product agent is the ConcreteSubject. Other companies that are interested in receiving location updates for a shipment can register as Observers with the ConcreteSubject. When the ConcreteSubject product agent receives a location update, it immediately forwards it to all registered Observer product agents. The solution has now been tested in two separate field installations for tracking project deliveries.

The product that is tracked can be linked to any number of observers, and require no prior knowledge of who these observers will be. This means that in this pattern the workflow does not need to be known beforehand, as long as the companies involved can install the middleware components needed, i.e. the client software for tracking checkpoints and the product agent component for creating new observers.

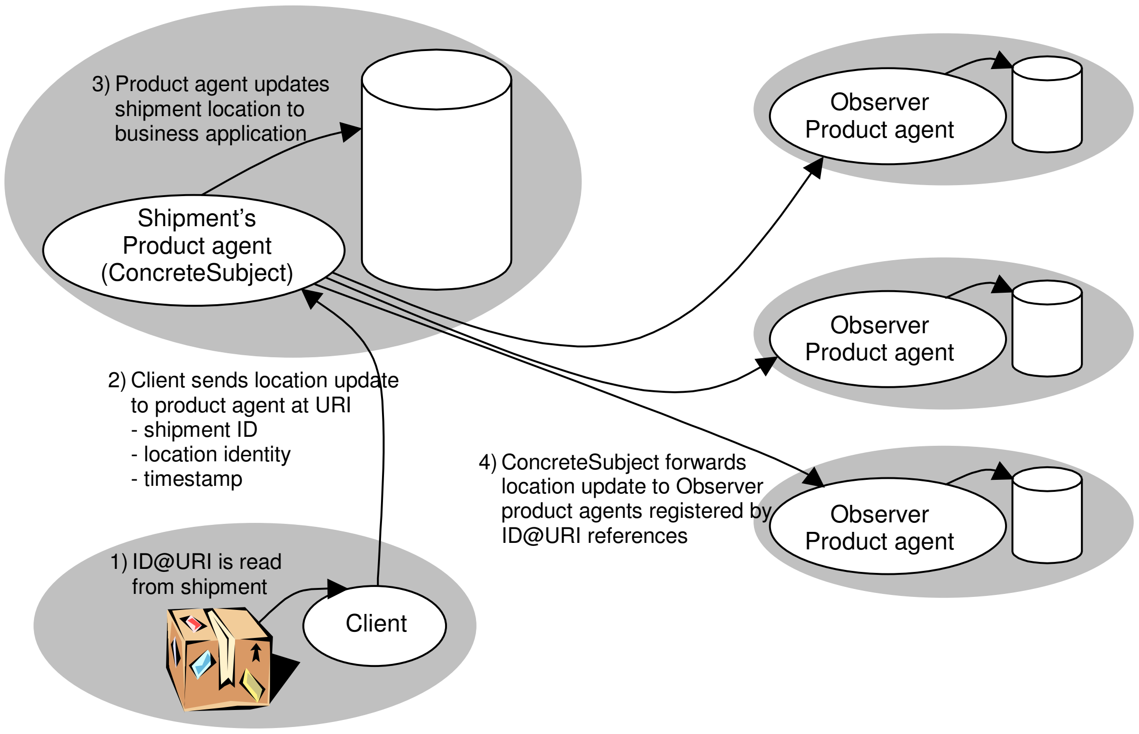
3) Product agent updates shipment location to business application
Shipment's Product agent (ConcreteSubject)
Observer Product agent
Observer Product agent
Observer Product agent
2) Client sends location update to product agent at URI
- shipment ID
- location identity
- timestamp
4) ConcreteSubject forwards location update to Observer product agents registered by ID@URI references
1) ID@URI is read from shipment
Client

*Figure 1: Loosely coupled track and trace*

*Track and trace of composite products*

The need for real time tracking and tracing of composite structures is found in the delivery of complex customized products and for project deliveries. Also other business situations where the delivery has a unique structure and is delivered by a network of suppliers need solutions to maintain delivery and inventory visibility. Today no such solutions exist outside the integrated single enterprise and a handful of key suppliers.

The difficulty to trace composite products arises because the structure of individual products is described by referring to a static bill of material (BOM) that is linked to the product type (or product class). This is the analogue to differentiation by class inheritance. Differentiating objects by class inheritance, or subclassing, is difficult because the design scheme is static (Gamma et al., 1995, p. 19). It is not possible to change the object features inherited from parent classes at runtime. For tracking and tracing composite products describing the product structure by class inheritance is not a good solution because different companies need to handle and perhaps modify the composite product in the delivery chain, i.e. "at runtime" of the customized delivery.

According to Gamma et al, (1995, p. 11) structural patterns have been developed in object-oriented programming to help decomposing a system into objects. The problem is to identify what design patterns would make it possible to track customised products and composite project deliveries across different organizations without having to change the loosely coupled tracking and tracing design of the previous section. The potential patterns to resolve this task of traceability of composite products are: Bridge, Chain of Responsibility, Composite, Decorator, Observer and Strategy.

The Composite pattern is the most promising one in this case. In object oriented programming this pattern is used for representing part-whole hierarchies of objects in situations when you want to ignore the difference between a composition of objects and individual objects. These conditions for when to use this design pattern fit well the problem situation of composite product structures and orders in logistics operations design. It has also been pointed out that equipment such as computers is modelled naturally using the composite pattern (Gamma et al., 1995, p. 170).

The participants of the Composite pattern are the Component interface and its implementing classes Leaf and Composite. Leaf objects have no children while Composite objects have children and associated methods (add, remove, get children). Component defines one or a set of operations that are applicable both for Leaf and Composite objects. For a composite product made from subassemblies of different companies, associating every subassembly with the assembled product will build up a composite data structure. In this situation the “get children” operation is already sufficient for ensuring complete traceability. If parts of the assembly are changed during maintenance, then this change can also be reflected in the corresponding data structure.

The Composite design pattern was implemented by adding messages to the product agent that had been developed for tracking and tracing shipments. The added messages are “add”, “remove”, “get children” and “get parent”. The message structure is explained in greater detail in (Främling, 2002) and (Främling et al., 2004).

Only object references are needed for managing the composite data structure. This means that only information on the parent-child relation and the two ID@URI records concerned need to be stored in a database (e.g. “part-of”;”ID1@comp1.com”;”ID2@comp2.com”). Such information can be handled by middleware software without making changes to existing business information systems.

The benefit of the Composite pattern for managing complex product structures is that primitive and complex objects can be handled the same way for many tasks. It offers a simple and universal solution to product traceability of physical composite products. The disadvantage is that the outlined pattern scheme needs runtime checks to enforce constraints on components, and that the composite structure has to exist physically because the identity is linked to the physical product.

### *Dynamic order structures*

In the previous section we studied how to associate product information to physical product item structures. However, similar information management requirements exist before the physical product item is manufactured and shipped. In order to manufacture a customised product it is necessary to define the product’s composition and the manufacturing operations that need to be performed. In the case of a virtual enterprise there is also a need to plan and specify what entity will deliver what material and/or perform what operation before work is started.

Introducing dynamic product and order structures would signify moving from a tightly coupled model to a loosely coupled model. Because we are again dealing with a part-whole hierarchy the Composite pattern again plays an essential role for a solution. However, a major difference between order structures and composite products is that no physical product items exist for order structures. So what kind of objects could we use as nodes of the composite hierarchy? The objects to use should preferably also be suitable for negotiating with potential performers who should perform the operation.

It turns out that addressing these requirements calls for the combination of several different patterns. The pattern selected for representing nodes of the Composite product and order structures is the Command pattern. Command encapsulates a request as an object that, in the case of orders, defines the properties of the product and the operations that need to be performed on it. You can assemble commands into composite commands and new commands can be added without changing existing ones

(Gamma et al., p. 237). Such a composite command structure is suitable for creating a dynamic order structure.

The Command design pattern enables many applications that are useful in inherently distributed operations environments. The Command pattern is also known as the Transaction pattern or Action pattern. The principle of the design pattern is to encapsulate the concept that varies (Gamma et al.,1995, p. 60). The command or transaction is made into an object that can be handled independently of the issuing entity. The design goal of the pattern is to issue requests without the need to know anything about the operation or the receiver of the request, and make it possible to more conveniently track, process, pass along, and undo requests. This way the pattern decouples the object that invokes the operation from the one having the knowledge to perform it.

For assigning the performer of an operation, for instance, all potential performers of the operation could register as Observers for the need of that operation. When there is a need for such an operation, the Command can be issued to all registered Observers then select the most appropriate is selected. The Command design pattern also makes it possible to track and trace requests back to the issuer, for example the end-customer. The pattern is also useful for dealing with suppliers and service providers that do not fulfil the requests adequately because the design scheme allows for requests being reversed ("undo" operation).

The functionality for the Composite part has been implemented as explained in the previous section. Using the Command pattern as described in this section requires that the ID part in ID@URI references should not be restricted to identifying only physical products. Globally unique ID@URI references also need to be created for purely virtual objects. This requires a mechanism for allocating unique ID's to and managing the identity of any kind of items. Such mechanisms exist already and are widely used in peer-to-peer (P2P) applications both for identifying nodes of the network (i.e. the URI part)and individual items, e.g. music or video files.

*Planning and scheduling of composite products and dynamic order structures*

The next source of redesign in a network operation is dealing with improved algorithms and implementations of objects. Introducing a new and better solution leads to redesign if there are algorithmic dependencies. There are many types of logistics operations and services that potentially need to access and use the data structures developed in previous sections, e.g. planning and scheduling of composite products and dynamic orders. In current supply chain systems, data structures and the algorithms that operate on them tend to be tightly coupled together.

However, there are many well-known examples of loosely-coupled algorithms, e.g. Internet search engines that go through the network of HTML links (i.e. references) of web pages on the Internet in order to produce an efficient search database. Another example is different verifier algorithms that are used in CAD systems for construction planning to verify the validity of the design (e.g. resistance, non-collision etc.). In the same way we can perhaps develop loosely coupled algorithms for operations planning and control that operate on the previously presented product information and order structures. These algorithms would be loosely coupled in the sense that it is easy to change the algorithm that is used. Typical loosely coupled logistics operations that would need to be performed in a changing participant network are checking for configuration errors, scheduling operations and selecting suppliers of the needed operations. The algorithms to use depend on the configuration of the product and on how changing the supplier network is. Therefore the algorithm to use may vary even from one product item to another in a composite structure. It is also most likely that new algorithms will replace existing ones as new requirements appear.

In object oriented programming the problem of switching algorithms on the fly is approached by using behavioural design patterns. The two design patterns that seem to be the most appropriate are

Strategy and Visitor patterns. The benefit of using these patterns is that behaviour can be changed without modifying the underlying data structure. The Strategy pattern defines a family of algorithms for a given purpose and encapsulates them in a way that makes them interchangeable. When using the Visitor pattern the data structure is given as a parameter to the algorithm, which means that it is the data structure that is programmed against an interface.

The Visitor pattern can be used to good effect when there are many distinct and unrelated operations to be performed on objects in a structure. The pattern is also useful when the object structure contains many classes of objects and you want to perform operations that depend on their concrete classes. These conditions hold for a great number of activities that are needed when using dynamic composite structures for products and orders.

Participants of the Visitor pattern are the Visitor interface, the ConcreteVisitor classes that implement the Visitor interface, the Element interface and the ConcreteElement classes that implement the Element interface. The Visitor interface defines methods for a set of related operations.. This means that it could for example define one method for planning the production of the engine in a car, one method for planning the production of the chassis, and another for planning the assembly. The Element interface defines an "accept" method that takes a reference to a Visitor as parameter. The "accept" method can check the capabilities of the Visitor before calling the actual method that performs the operation. The use of "accept" may seem overly complicated but makes the pattern more general because it lets the visited element check that the visitor is indeed capable to perform the required operation. An example of this is that an element in an order structure could be programmed to accept a production-planning visitor but not a product maintenance visitor because the physical product has not been manufactured yet.

This pattern has not been implemented in the pilot and test environment under development. However, as shown by search engines on the Internet and similar applications, the Visitor pattern is powerful when data is structured according to standards.

## CONCLUSION

The loosely coupled operations designs studied here using well-known patterns from object-oriented programming result in a pattern sequence for implementing a loosely coupled virtual enterprise environment. The resulting pattern sequence is summed up in table 2. The table lists the logistics operations design, the design pattern, and empirical evidence of the relevance and functionality of the operations design.

*Table 2: Pattern sequence for implementing a loosely coupled virtual enterprise environment*

| **Operations design** | **Selected pattern** | **Empirical evidence** |
|---|---|---|
| Shipment track and trace | Observer | Functional prototype system design<br>Field trials in two project delivery networks |
| Track and trace of composite product | Composite | Functional prototype system design<br>No field trials yet |
| Dynamic order structure | Command | --- |
| Planning and scheduling of composite products and dynamic orders | Visitor | Functional analogy to Internet search engines and analysis services for electronic product models |

The analysis indicates that track and trace based on the observer pattern and dynamic object composition using the composite pattern are the cornerstones for implementing loosely coupled operations in evolving multi-company networks. These two basic pattern designs are preconditions that need to be in place for dynamic orders and evolvable planning and scheduling services.

The benefit of using the Command pattern for describing dynamic order structure is that the order structures can be easily manipulated by other patterns such as Observer, Composite and Chain-of-Responsibility. The immediate business benefit comes from easier handling and traceability of design and manufacturing information related to customized products.

The benefit of using the Visitor pattern is that it introduces loose coupling between data structures and algorithms that work on them. The business benefit comes from the possibility to use third-party algorithms (e.g. planning and scheduling modules, verification modules) more easily and replace old algorithms with more efficient ones without modifying the rest of the system, thereby improving the quality of design, production and products.

The pattern-based object schemes presented in this paper enable the path-based approach that Hayes et al. (2005, p. 185) have called for in operations management. It does not demand that all solution components be specified, or even anticipated, in advance. But as information and processing capability becomes an increasingly important part of products, companies that can provide these capabilities also become an increasingly important part of the virtual enterprise networks. How companies organize and handle product-related information will determine how well they will be able to make the transformation to customized products and services, and a virtual enterprise environment.

The work also contributes to Computational Organization Theory (COT). COT uses mathematical and computational methods to study both human and automated organizations as computational entities (Carley and Gasser, 1999, p. 4). The use of design patterns in this paper is an example of developing the building blocks for a computational distributed organization.

## ACKNOWLEDGEMENTS

The authors wish to thank the Academy of Finland and the National Technology Agency (TEKES) for supporting the research.

## REFERENCES

Alexander, C. (1977), *The Timeless Way of Building*, Oxford University Press, New York

Carley, K. and Gasser, L. (1999), "Computational Organization Theory", Chapter 7 in Gerhard Weiss (ed.), *Distributed Artificial Intelligence*, MIT Press

Davenport, T. H. and Short, J.E. (1990), "The new industrial engineering, information technology and business process redesign", *Sloan Management Review*, 31 (4), pp. 11-27

Davis, M. (2003), "Editing Out Video Editing", *IEEE Multi Media*, 10 (2), pp 2-12

Främling, K., (2002), "Tracking of material flow by an Internet-based product data management system (in Finnish: Tavaravirran seuranta osana Internet-pohjaista tuotetiedon hallintaa)", Tieke *EDISTY magazine*, No. 1, 2002, Publication of Tieke (Finnish Information Society Development Centre)

Främling, K., Kärkkäinen, M., Ala-Risku, T., and Holmström, J. (2004), "Managing Product Information in Supplier Networks by Object Oriented Programming Concepts", In: *Proceedings of IMS International Forum*, Cernobbio, Italy, 17-19 May 2004; pp. 1424-1431. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995), *Design Patterns*, Addison-Wesley Professional Computing Series, Reading, Massachusetts, USA, !995, pp.395

Gausemeier, J. and Gehnen, G. (1998), "Intelligent material flow by decentralized control networks", *Journal of Intelligent Manufacturing*, vol. 9, pp. 141-146

Hammer, M. (1990), "Reengineering Work: Don't Automate, Obliterate", *Harvard Business Review*, Jul/Aug 1990. Vol. 68, No. 4; pp. 104-112

Hayes, R, Pisano, G., Upton, D., and Wheelwright, S. (2005), *Operations, Strategy, and Technology – Pursuing the Competitive Edge*, John Wiley & Sons, Hoboken, NJ

Holmström, J, Främling, K., Tuomi, J., Kärkkäinen, M., and Ala-Risku T. (2002), "Intelligent product agents: The key to implementing collaboration process networks?", *International Journal of Logistics Management*, Vol. 13, No. 2, pp. 39-66

Kärkkäinen, M., Holmström, J., Främling, K., Artto, K. (2002), "Intelligent products - a step towards a more effective project delivery chain.", *Computers in Industry*, vol. 50, no. 2., pp. 141-151

Kärkkäinen, M., Ala-Risku, T., Främling, K. (2003) "The product centric approach: a solution to supply network information management problems?", *Computers in Industry*, vol. 52, no. 2, pp. 147-159.

Otto, A. (2003), "Supply Chain Event Management: Three Perspectives", *The International Journal of Logistics Management*, Vol. 14, No. 2, pp. 1-13

Petrie, C., Goldman, S, Raquet, A. (1999), "Agent-Based Project Management", *LNAI*, vol 1600, Springer-Verlag, 1999, http://www-cdr.stanford.edu/ProcessLink/papers/DPM/dpm.html

Petrie, C. and Bussler, C. (2003), "Service Agents and Virtual Enterprises: A Survey", *IEEE Internet Computing*, July/August 2003, pp. 68- 78

Simon, H. (1996), *The Sciences of the Artificial*, The MIT Press, Cambridge, Mass., USA, 3rd Edition, pp. 231

Töyrylä, I., (1999), *Realising the potential of traceability – A case study research on usage and impacts of product traceability*, PhD. Thesis, Acta Polytechnica Scandinavica. Ma, Mathematics, computing and management in engineering series ; 97, Finnish Academy of Technology, Espoo.