73 PROVIDING TRANSPARENCY IN THE BUSINESS OF SOFTWARE: A MODELING TECHNIQUE FOR SOFTWARE SUPPLY NETWORKS

Slinger Jansen
s.jansen@cs.uu.nl, Utrecht University, NETHERLANDS
Sjaak Brinkkemper
s.brinkkemper@cs.uu.nl, Utrecht University, NETHERLANDS
Anthony Finkelstein
a.finkelstein@cs.ucl.ac.uk, University College London, UK

One of the most significant paradigm shifts of software business is that individual organizations no longer compete as single entities but as complex dynamic supply networks of interrelated participants that provide blends of software design, development, implementation, publication and services. Understanding these intricate software supply networks is a difficult task for decision makers in software businesses. This paper outlines a modeling technique for representing and reasoning about software supply networks. We show, by way of a worked case study, how modeling software supply networks might allow managers to identify new business opportunities, visualize liability and responsibilities in a supply network, and how it can be used as a planning tool for product software distribution.

1 SOFTWARE BUSINESSES ARE BLENDS

Individual businesses no longer compete as single entities but as supply chains (Lambert, 2002). This holds for the software industry as well, where software products and services are no longer monolithical systems developed in-house, but consist of complex hardware and software system federations (Ghezzi, 2002) produced and sold by different organizations. This development has led organizations to combine their business and components into complex software supply networks (SSNs), from which they supply end-users with integrated products. As these SSNs grow more complex, it becomes harder for the participants of SSNs to make informed decisions on development strategy, responsibility, liability, and market placement (Gartner, 2005; Grieger, 2003). It also becomes harder to manage the risk associated with these decisions (Jansen, 2006A).

A SSN is a series of linked software, hardware, and service organizations cooperating to satisfy market demands. SSN management is different from physical goods supply chain management (SCM) in two ways. First, software is malleable after release and delivery, giving rise to the need for extensive maintenance. Secondly, products delivered to end-users in SSNs are tolerated with much lower quality levels than other products (Baxter, 2001). A result of the difference between conventional supply networks and SSNs is that literature on collaboration in supply networks (Patosalmi, 2003) does not discuss maintenance and how it requires information about the supply chain. The same holds for other work on SCM, such as (Lazzarini, 2001), which groups horizontal ties between firms (such as manufacturers

Jansen, S., Brinkkemper, S., Finkelstein, A., 2007, in IFIP International Federation for Information Processing, Volume 243, Establishing the Foundation of Collaborative Networks; eds. Camarinha-Matos, L., Afsarmanesh, H., Novais, P., Analide, C.; (Boston: Springer), pp. 677–686.

and suppliers), but fails to recognize the importance of leveraging feedback in such networks, or Lambert and Cooper (Lambert, 2002), who provide a conceptual framework for SCM without maintenance.

This paper explores the new field of SSN research by presenting a method for modeling the complex relationships between participants in the supply networks of composite products and services. By conducting a case study of an organization that leverages the SSN we demonstrate that SSN models enable participants in supply networks to reason about business identification, product architecture design, risk identification, product placement planning, and business network redesign. Furthermore we demonstrate that modeling relations in supply networks is the first step in explicitly managing relationships with other participants.

Value chains differ from SSNs in that value chains describe one product only, whereas SSNs specifically address networks of software systems that interact to provide software services. Attempts have already been made to model value chains surrounding large ERP configurations, by Messerschmitt and Szyperski (Messerschmitt, 2003). Their model cannot represent relationships between for instance a component-off-the-shelf (COTS) vendor and an application developer, making their models insufficient to describe a complete SSN. Weill and Vitale's (Weill & Vitale, 2001) value chains describe supply networks best, but lack the accompanying product description required for a software supply network model.

In the following section SSN models are provided, by presenting the participants, the meta-model, a creation method, and an example. In section 3 the case study Tribeka is presented. In section 4 applications of SSN models are presented and discussed. In section 5 a description is provided on how organizations best ready themselves to participate in a supply network.

2 SOFTWARE SUPPLY NETWORK MODEL

Models for SSNs consist of two parts, a product context and a supply network. A **product context** describes the context, in which a software service operates, and the software products, hardware products, and software services that are required to provide the software service. A **supply network** displays all participants in a SSN, the connections between these participants, and the flows describing the type of product that is traded across these connections. **SSN Participants** are any party that provides or requires flows from another participant in the network. The two diagrams are complementary in that the product context shows all products that are traded in their different forms in the supply network.

2.1 Product Context

A software service is the provision of one or more functions by a system of interest to an end-user or to another software service. A software system is a combination of independent but interrelated software services, software components, and hardware components that provides one or more services. There are three types of entities in the product context, being (1) white-box services and their systems, (2) black box services, and (3) software and hardware components making up systems. The main concept for product contexts is that of a software system, which consists of hardware and software components and when combined and activated these components provide a service. A software system requires at least one hardware component, at

least one software component, and any number of services. A hardware component can support any amount of services and software components.

When a software component requires other software components (such as libraries and COTS) these are drawn under the software component. When a software component requires software services (such as databases and web servers), these are drawn under the software component as a service. The hardware on which the software components are running is drawn left of the systems. It is assumed that hardware components are complete, and thus do not have dependencies. Systems that provide services are drawn as containers containing the software components and services on which they depend. Please see Figure 2 for an example.

2.2 Supply Network

Supply networks connect participants in the network. Furthermore, these connections are annotated with flows, such as product requirements, product designs, software components, component assemblies, assembled products, assembled deployed systems (hardware and software), and services (provided by these systems). These artifacts all come from decoupling points for software products (see Figure 1), which is the point at which demand and supply meet in a supply network.

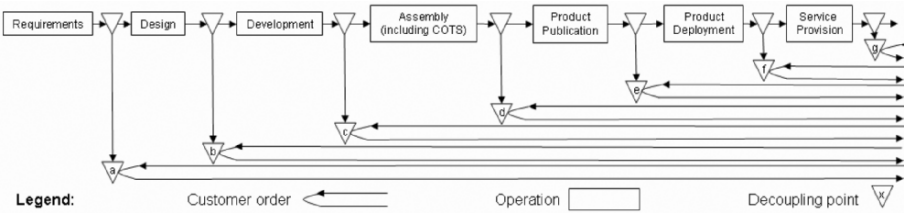

Figure 1 Product Software Decoupling Points

To date, product software is defined as a packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market (Xu Brinkkemper, 2005). When looking at the product software production pipeline seven decoupling points can be identified. First, a development organization can outsource the requirements engineering process and/or design process (*a, b*). Also, the developer can choose to release their source code, binaries, or assemblies of components (*c, d, e*) to another developing organization who uses these artifacts as a component to their product, or to a publisher who releases the product (common for games, where the vendor is rarely the developer). A software vendor can also choose to release the product itself, either as a package, or as a deployed system (*f*). Finally, a vendor can decide to offer their product to its customer in an application service provider model, where the vendor sells usage of its product instead of the product itself (*g*).

Flows are modeled as labels on the arcs between the participants and are distinguished by different colors and codings. The color indicates whether the artifacts are source artifact collections, compiled binary artifact collections, or complete systems and services. The codings are (in order of creation to usage) **Req** (requirements), **Des** (design), **Comp** (software component), **As** (software component assembly), P (software product), Sys (system, including hardware), HW (hardware),

and finally Ser (services). It is not uncommon for software products going through iterations of the decoupling points before the product is delivered to a customer. It can be well imagined that a system designer creates a design, sells the design, and the software developer starts at the requirements phase again to see what can be added to the design. To indicate this numbers are used in the codings after the abbreviation, such as Des2.1, which means that this is the second design for product 1. In the supply networks we only make this distinction when two generations of artifacts are produced by different participants.

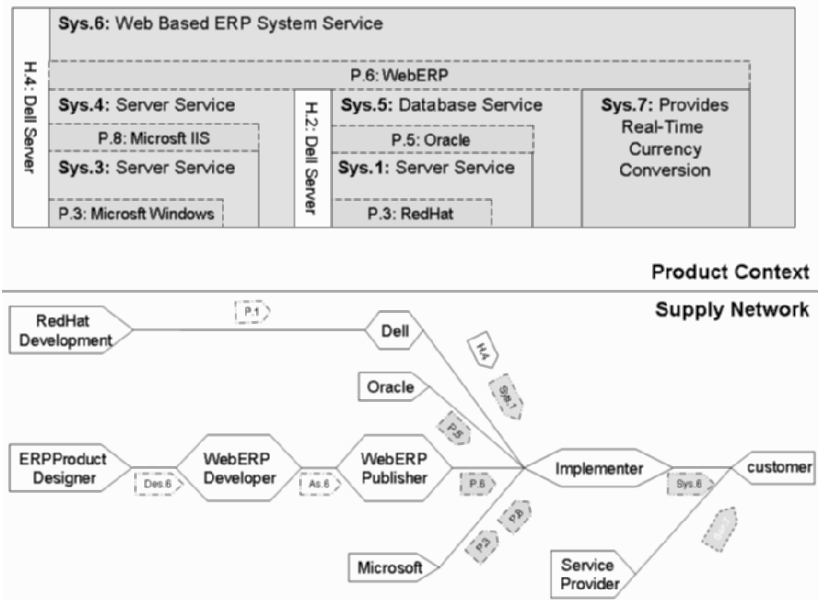

Figure 2 Example of a WebERP SSN Model

In Figure 2 the example models are presented for a customer requiring a Web Enterprise Resource Planning (ERP) service. To supply this service internally, the customer has decided to go with its personal implementer organization who implements a product WebERP on a newly purchased local database server and a local web server. The product context displays that to supply Ser.6, P.6 is required. To run P.6 a server is required that supplies WebERP through a web server application, in this case Microsoft IIS. On the other side a database server (Sys.5) is required that manages all the data for WebERP. Both servers, supplied by Dell, run a different operating system. Furthermore, to provide the WebERP service, a currency conversion web service is required. As products transition from source code to product to system, they generally retain the same number, such as for WebERP; Des.6, As.6, P.6, and Sys.6 are all instances of the same (software) artifacts sold at their different decoupling points.

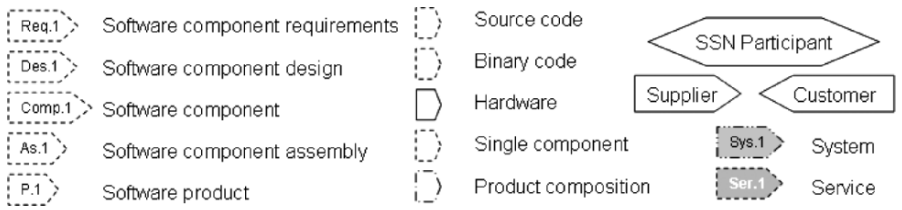

Figure 3 Supply Network Legend

2.3 SSN Model Creation Method

To help define the scope of a SSN model, the product context is created. The product context, which describes the systems that supply software services, must display all products and services that are specifically required for the service(s) of interest. Secondly, the participants must be determined. These participants are all parties that are involved with the products in the product context. The products in the product context will be presented as flows later. Finally, all relationships must be established between the participants. Whenever a product or service is traded between participants, they must be connected by an arc. Once the arc is drawn, it can be annotated with the type of products and services that flow down this arc. Please note that there must be a strict consistency between products in the supply network and the product context. Furthermore, flows are directional, such as software flowing forward to customers and money or feedback flowing back to vendors.

3 A CASE STUDY: TRIBEKA

We use a case study to demonstrate the SSN modeling technique. The company under study is Tribeka (http://www.tribeka.com), an organization that attempts to break through the traditional product software retail supply chain, by delivering assemblies of components to retail outlets that can be burnt, packaged and turned into a finalized product *on-site*. Tribeka, founded in 1996, currently employs twenty five people and has deployed its systems at large retail chains in the United Kingdom, such as WH Smith and HMV. Recently Tribeka has opened four high street outlets where it solely sells software created with Tribeka's SoftWide system.

The Tribeka SoftWide system consists of a server with a large storage memory, an internet connection, a number of CD and DVD burners, and a high quality cover printing facility for boxes, CDs, and DVDs. It is capable of creating between 50 and 100 different shrink-wrapped products per hour. The SoftWide system is not solely a hardware solution, since it is able to deliver the most up-to-date software onto the retail market. On a daily basis software updates are sent to the SoftWide systems deployed in retail stores, including price information. The SoftWide system stores the component assemblies in a coded manner, such that products are only produced when requested and authorized, using a proprietary auditable licensing system.

3.1 Tribeka Models

An SSN model with two different supply networks is presented in Figure 4. At the top level of the figure the product context displays two systems, that provide the

"computer use" and "entertainment" services. The entertainment system requires the software service "computer use" and the game product **P.3**. The system **Sys.2** requires a laptop and Microsoft Windows, before it can actually provide the "entertainment" software service.

Traditional software supply is depicted in the ``before Tribeka'' section of the figure. Here Microsoft is modeled as a software developer, who delivers its product to Dell. Dell, the hardware manufacturer, deploys the product **P.1** onto the laptop system and delivers **Sys.2** to its retailer, PC Store. PC Store sells the system to the customer, who also purchases a game **P.3** with it. **P.3** is designed by Game Designer and the design is sold to the Game Developer, which actually implements the game. Once game development is finished a collection of source components (**As1.3**) is sent to the game publisher. These source components are then compiled, causing the **As2.3** to be shaded, and sent off to a printing facility. Finally, the game publisher sells the finished products **P.3** to a reseller. The reseller then sells the game to PC Store.

Tribeka takes over from the Game Publisher, the Reseller, and the printing facility, by directly publishing any product from a software developer to retail stores. The Game Developer now passes a compiled set of components directly to Tribeka. Tribeka sends the component assembly to PC Store directly, instead of to a printing facility and then reseller. The component assembly is then assembled into a product at the retail store, enabling the latest possible binding for physically sold software products. Tribeka also has opened three retail stores itself, offering all products offered through the SoftWide system.

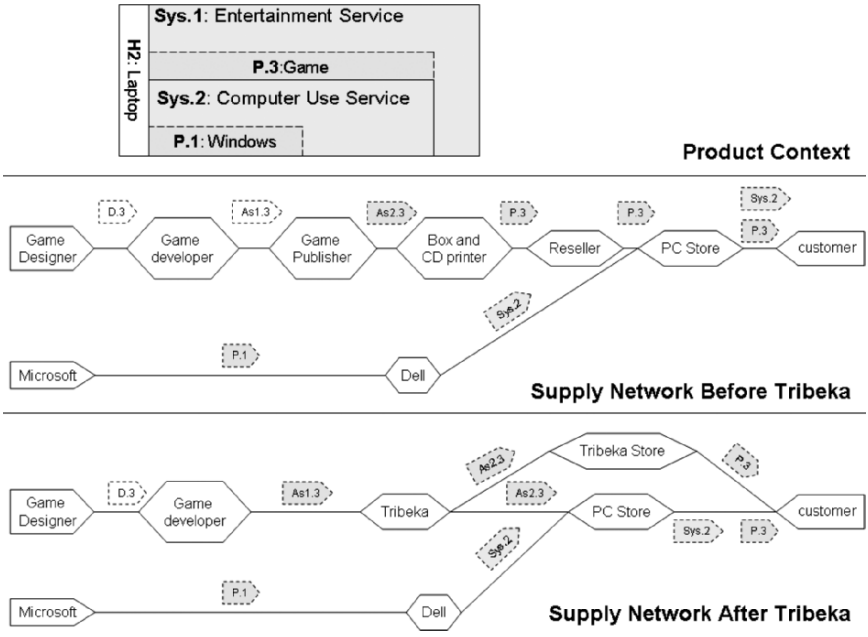

Figure 4. Tribeka Case Study SSN and Product Context

3.2 Tribeka Relationships

The SSN model in Figure 4 shows that Tribeka maintains intensive relationships with the game developer and with retailers. The presented model is slightly simplified because in many cases there will be a publisher in between the developer and Tribeka. As such, Tribeka has three types of participants in the supply network it deals with: retailers, game developers, and game publishers. Tribeka uses its SoftWide system to maintain relationships and transport data between these participants. Publishers and developers send their component assemblies to Tribeka, which are then uploaded into the SoftWide system, including price information, license codes, software artifacts, digital manuals, and images for box covers. These publishers are able to see the status of their products, such as how many sales have been made and what types of licenses have been distributed. On the other side retailers access the SoftWide system through their points of sale, which are used to sell and create software products from the product assemblies supplied by Tribeka.
From Tribeka two lessons can be learnt about SSNs. To be successful in a supply network an organization must explicitly manage relations with the other participants. The second lesson is that an organization must observe opportunities to take part in different parts of the supply network, such as Tribeka's opening of retail stores that only use the SoftWide system.

4 SSN MODEL APPLICATIONS AND USAGE

We have identified five applications of SSN models being business identification, product architecture design, risk identification, product placement planning, and business network redesign. The aim of SSN models is to clarify the blend that is software business. SSN models are thus used by policy makers, software architects, and entrepreneurs. Depending on the application, they must make the SSN model on a regular basis and observe changes, risks, and opportunities. The SSN model can function as an overview diagram for a business plan or even for year end-reports to indicate how a software business made profit.

Business Identification - SSN models show the trade relationships for each participant in the network. These flows can be used to determine the business type for that participant. When, for instance, a participant receives hardware and software components and has one system as output this is an *integrator* (Implementer in the WebERP example). A participant that receives component assemblies and then publishes products is a *publisher* (WebERP publisher in the WebERP example). Another common example is a supplier that has no input but produces a software product (*software product developer*, RedHat in the example). We see that Tribeka (see Figure 4) functions as a *packager* and interestingly enough turns the PC Store into a *software product publisher*. A participant that has the same input as output is a *reseller*. Finally, according to these definitions and due to the absence of a hardware component input Dell is a *hardware producing integrator*. These constructs are commonly encountered in different SSN models.

Product architecture design - In deciding the type of software architecture a software developer must use, the supply network plays an important part. The software architecture decides how a product will depend on other products and services, and this will have far reaching consequences on the future of a software product. SSN models can thus assist in making architectural design decisions.

Risk identification - The SSN model uncovers, for instance, that a product cannot be used without the availability of some component or service. These dependencies on other organizations, though logical, can be disastrous for participants further up the supply network. Such a dependency influences the total cost of ownership of the product, the possibility to internationalize, and even the future when such a dependency can no longer be fulfilled. This calls for diversification and architecting for product dependency variability (Jaring, 2004). The SSN model helps uncover such relations and dependencies. SSN models can be used by customer organizations to uncover whether they are in possession of certain products that are unsupported, or whether they are making use of a service that could easily be terminated. Such investigation is part of portfolio rationalization. A common vulnerability, for instance, is a custom link between two products, built by a software implementation service provider, which stops working after an update for one of the products has been deployed. The SSN models can assist in finding and eliminating such weaknesses for all participants in the supply network.

Product placement planning - A vendor can use the SSN model to determine how to market its product, how to inform customers of product news and releases, and how customers will contact the vendor. The latter is especially important when looking at pay-per-usage feedback and error feedback (Jansen, 2006b). For example, when a bug report is sent from a customer to a participant in the SSN the participant must decide whether to solve this issue or to forward it to another party in the supply network. Software vendors can choose to sell their products as add-ons to other products, in combination with hardware (i.e., navigation systems), and as a service (on-line bookkeeping). Furthermore, software vendors can decide to sell the product through channels they own (their own site), through resellers, through service providers, etc.

Business Network Redesign - Participants of the supply network must identify their business partners and establish different types of relations. SSN models can thus be used to design business information systems that take into account the participants of the supply network with which the business will have regular and even ad-hoc relations. Tribeka for example manages explicitly its relationships with software developers, publishers, and retail outlets and has created different information systems and portals for them.

The SSN model reveals business opportunities and risks by making two types of changes. The first change is to alter the binding times and decoupling points for products and services. Tribeka is a clear example of this, where it takes the role of the traditional reseller, but simply assembles the product at a later stage. This optimization allows resellers to replenish their stock dynamically, saving cost in the area of stock management, delivery, and deployment. The second type of change is seen when a change is made to the participants in the supply network, in the form of acquisitions, split-ups, developers buying new COTS, or customer organizations that become vendors of products or services themselves. An example of this is when Tribeka opens their own SoftWide stores.

5 AD-HOC SOFTWARE SUPPLY NETWORKS

The tendency to integrate components from different developers and manufacturers into new products and components by both customers and integrators has led to a phenomenon of quickly forming and dissolving of ad-hoc supply networks (Zager,

2000). Many organizations, however, are not specifically adjusted to manage relations within such ad-hoc networks. Simultaneously, software vendors and manufacturers are constantly approached by (new) business partners, such as manufacturers, resellers, and service providers, with bugs, feedback, requests for changes, and other communication about their software products.

A coalition between participants in a supply network is where participants rely on each other, yet do not have any of the skills required for collaborative unity (Zager, 2000), such as organizational measures, structured communication, and planned durability. Software organizations can profit from the many opportunities in these ad-hoc supply networks when properly prepared to engage (order of intensity) in conversations, relations, partnerships, and even alliances with other participants. These other vendors are willing to create a user and developer community around a (configuration of) software product(s), which will encourage use of products and create new solutions and opportunities surrounding them. An example of such a relationship between software vendors is when Microsoft sends error messages to product vendors whose products have crashed on Windows. The vendor can opt to resolve the error independently or in different gradients of intensity with Microsoft.

The SSN modeling technique presented in this paper assists a participant in understanding how these coalitions are formed. Secondly, a participant must build a community surrounding its product that unifies external and internal users, developers, implementers, and integrators of the product. Such a community can be built using ontologies, portals, customer days, and partner days. Especially portals, which can be used for the distribution and sharing of knowledge, development and bug finding tools, are an important factor to create a close network of participants willing to add value to the community, and thus increase value of a supply network.

To transform a coalition to collaboration relations must be formalized by the facilitating organization. A clear distinction needs to be made between intensively and loosely coupled alliance partners. By classifying partners in such a way, participants can create different circles of trust with partners and users, which will clarify the different relationships within a supply network considerably for all participants in a network. A participant in the supply network must at all times be aware of opportunities to form coalitions, since each customer question could set in motion the cooperation between multiple participants. The belief that supply networks must be leveraged by software businesses is further strengthened by (Duyster et al., 1999) who claim that to craft successful strategic technology partnerships steps need to be undertaken to strategically position participants, prepare alliance skills, build a business community, and do smart partner selection.

6 CONCLUSIONS AND FUTURE WORK

SSN models provide a novel manner to perform strategic and risk evaluation in the business of software. The Tribeka insights can be realized through experience, but the ability to assess risks a priori is a valuable contribution. This paper presents a modeling method for SSNs to provide insight into supply networks, enabling participants to do risk assessment, strategic decision making, product placement planning, and liability determination. A case study is used for this paper because it is a proven method to introduce a novel research area, such as SSNs.

The decoupling points are a concept taken from physical product development and marketing planning. The combination of supply network with these decoupling

points creates a powerful modeling tool that is generalizable to non-software products as well. To be able to do so one must define the decoupling points, the possible range of product decompositions, before creating the supply networks.

Currently we possess a collection of 30+ SSN models from start-ups and medium to large software enterprises. With these models we are hoping to further classify different business models for product software. Furthermore we are experimenting with different flows, such as content, money, and licenses.

7 REFERENCES

1. Baxter, L. and Simmons, J. The software supply chain for manufactured products: reassessing partnership sourcing. In International Conference on Management of Engineering and Technology, 2001.
2. Colville, R. and Adams, P.. It service dependency mapping tools provide configuration view. In Gartner Research News Analysis. Gartner, 2005.
3. Duysters, G., Kok, G., and Vaandrager, M. Crafting successful strategic technology partnerships. In Research and Design management, issue 4:29:343, 1999.
4. Ghezzi, C. and Picco, G. P.. An outlook on software engineering for modern distributed systems. In Proceedings of the Monterey workshop on Radical Approaches to Software Engineering, Venice (Italy), October 8-12, 2002.
5. Grieger, M, Electronic marketplaces: A literature review and a call for supply chain management research, In European Journal of Operational Research, 2003
6. S. Jansen and S. Brinkkemper. Definition and validation of the key process areas of release, delivery and deployment of product software vendors: turning the ugly duckling into a swan. In proceedings of the International Conference on Software Maintenance, September 2006b.
7. S. Jansen and W. Rijsemus. Balancing total cost of ownership and cost of maintenance within a software supply network. In proceedings of the IEEE International Conference on Software Maintenance, Philadelphia, PA, USA, September, 2006A.
8. M. Jaring and J. Bosch. Architecting product diversification - formalizing variability dependencies in software product family engineering. In Fourth International Conference on Quality Software, pages 154–161, Washington, DC, USA, 2004. IEEE Computer Society.
9. Lambert, D. M. and Cooper, M. C., Issues in supply chain management. In Journal of Industrial Martketing Management, 2002.
10. Lazzarini, S. G., Chaddad, F. R. and Cook, M. L.. Integrating supply chain and network analyses: the study of netchains. In Journal on Chain and Network Science. Wageningen Academic, 2001.
11. Messerschmitt, D. G. and Szyperski, C. Software Ecosystem: Understanding an Indispensable Technology and Industry (Chapter 6: Organization of the Software Value Chain. MIT Press, Cambridge, MA, USA, 2003.
12. Patosalmi, J. Collaborative decision-making in supply chain management. In Seminar in Business Strategy, 2003.
13. Weill, P., Vitale, M., "Place to space: Migrating to eBusiness Models", Harvard Business School, 2001
14. Xu, L., Brinkkemper, S., "Concepts of Product Software: Paving the Road for Urgently Needed Research", First International Workshop on Philosophical Foundations of Information Systems Engineering, LNCS, Springer-Verlag, 2005
15. Zager, D. Collaboration on the fly. In AIWORC '00: Proceedings of the Academia/Industry Working Conference on Research Challenges, page 65, Washington DC, USA, 2000. IEEE