Digital Business Ecosystem

Contract n° 507953

WP 13: Business networks

D13.2: Platform case studies and sensitivity analysis

Information Society Technologies

Project funded by the European Community under the "Information Society Technology" Programme

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: D13.2

Due dates: 10/2005

Delivery Date: 01/2006

Short Description:

Based on the previous report, concrete historical case studies are presented which teach important lessons about the success factors in establishing a software platform technology such as the DBE infrastructure, and the different phases a platform ecosystem goes through. The game theoretical analysis introduced in the previous report is extended to analyze the sensitivity to the global cost/revenue level, the technology structure, anti-trust regulations and asymmetry of vendor technologies.

Author: FZI

Partners contributed: FZI, UniS, STU (review)

Made available to: Public

Versioning

Version	Date	Author, Organisation
1.0	04/01/2006	Tim Romberg, FZI

Quality check:

1st Internal Reviewer : Konstantinos Giannoutakis (University of Surrey)

2nd Internal Reviewer: Thomas Kurz (Salzburg Technical University)

Contents

1 Introduction

This is the second report of the *business networks* work package of the Digital Business Ecosystem project. It strongly builds on the previous report *Economic models of software ecosystems*. The larger research question behind this work package is: *How are technology ecosystems based on a platform successfully created and sustained?* This report extends several parts of the previous report; the whole will eventually be integrated into one document. In the following chapter, we probe a little bit deeper into the origins of the ecosystem concept and its applicability or non-applicability to technology networks. Chapter 3 describes the cases studied during this activity, which were selected as major, well-documented technologies with a definitive "platform" value proposition. The following chapter presents the qualitative conclusions from these studies and the last chapter continues the theoretical analysis on technology networks already presented in the previous report.

2 Ecosystems

An ecosystem, in ecology, is defined as the combination of a biotope and the biocoenosis — the community of organisms of different species — populating it. There may be several reasons for the increased metaphorical use of the term ecosystem in the context of Information Technology, and several different implications intended by those who use it. This section looks at the context in which the term is used in ecology and potential applications to the IT world.

[MS03, Notes to chapter 9] justifies the analogy as follows:

> A biological ecosystem is an oft-used and reasonably good analogy because its dynamics depend on a complex mix of complementary and competitive relationships among all organisms. Like the software industry, pairs of organisms often share complementary and competitive relationships at the same time. For example, a parasite depends on its host but also detracts from it.

2.1 Species relationships in a biocoenosis

When two species populate the same habitat, their population densities can interact in one of the following ways[1]:

[1]The definitions of biological concepts in this and the following sections are taken from [CR01, Part 8 "Ecology and Behavior"], Wikipedia: "Biological interaction"

- (+/+) *Mutualism* – where both species benefit from the presence of the other;
- (+/-) *Predation* – where the first species benefits and the second species suffers from the presence of the other;
- (-/-) *Competition* – where both species suffer from the presence of the other;
- (+/0) *Commensalism* – one species benefits, the other is not influenced;
- (-/0) *Amensalism* – one species suffers, the other is not influenced;
- (0/0) *Neutralism* – no influence either way.

While there is consensus about these possible relationships among biologists, it is controversial whether individual interactions are strong enough to meaningfully influence the population densities in real biotopes, which typically have many more than just two species. In many cases, such as for plants, the population density for each individual species can be predicted quite well by only taking into account abiotic factors, such as humidity and temperature. For animals, [Wal92] suggested that species are mostly *redundant*, i.e. when one species, e.g. a certain predator, disappears, its place and function are quickly taken by one or several other species, such that there is no meaningful effect on the overall population densities.

There are certainly parallels to such relationships in economics and especially in the IT world. For many years, the press spoke of the *Wintel* alliance, a mutualistic relationship between Microsoft and Intel, which extended beyond any contractual obligations between the two very different companies. There is also, as shown later, generally a mutualistic relationship between a platform vendor and application developers. Competition is a very common relationship, be it competition for clients, for resources, or for personnel.

In order for the analogy to work in these two cases, "species" must be understood as "individual company", and biological "population density" as size or profitability of the company. One could also be tempted to understand "species" as company type or company business model, and "population density of species X" as the number of companies of type X. It would then also be possible to identify interactions between these numbers, but the problem is that a company is generally not interested in maximizing the number of other companies with the same business model as its own.

"Predation" has a different sense in the economic context, where it means the acquisition of one company by another company. This acquisition can be good for both, and so it does not correspond to the predation relationship in biology. A common, non-trivial question in the event of a merger or acquisition is: Who will benefit, who will suffer from this event?

Competitors of the merging companies may benefit if economies of scale are not large, because not all clients and employees of the merging companies may be satisfied with the new united leadership.

In IT markets, which are characterized by complex complementarity and substitutivity relationships, knowing who are friends and who are foes is a non-trivial problem in general. But just as in many biological ecosystems, eliminating today's foes may be costly strategy with only short-term effect, because other players can easily fill the gap - another example of redundancy. From a global point of view, redundancy may be a desirable property of both biological ecosystems and markets. Companies and business models can suddenly fail because of regulatory changes or scandals, and a healthy business ecosystem will be able to cope with such shocks quite quickly. In these regards, the analogy works.

On the other hand, there are several aspects of a business relationship which have no equivalent in biology: Companies can make enforceable contracts with one another; they can have several lines of businesses. They can shift their focus from one line of business to another, thereby changing the effects they have on other companies. When companies have common interests in one area, and compete in another area, this leads to a complex *coopetition* relationship ([BN96]). One example is the relationship between IBM and Microsoft, with close collaboration in the areas of Web Service standards and video consoles, and fierce competition in the areas of databases and workplace collaboration suites.

2.2 Food chain, energy flow and chemical cycles

Ecologists often study the energy flow and chemical cycles of an ecosystem. Energy enters an ecosystem usually in the form of sunlight, is stored and transformed in several steps and is lost in each step in the form of heat. The flow of energy is determined by the structure of the food chain (or: food web) of the ecosystem. The energy flow in the whole ecosystem is determined and restricted by the productivity of the primary producers, like plants and phytoplankton. As a rule of thumb, about 10% of the energy consumed by one level in the food chain reaches the next level in consumable form. From this rule, it is possible, for example, to estimate the population densities of organisms on lower levels of the food chain which are necessary to support a given population density of high-level predators, such as polar bears.

A similar analysis is possible with regards to essential nutrients in a food chain. Many ecosystems are limited in their productivity by one or very few such nutrients, while other nutrients and sunlight are abundant. In oceans, nitrogen or iron is often the limiting elementary nutrient.

This kind of analysis bears resemblance with the use that especially big IT companies often make of the word "ecosystem". Just as a polar

bear cannot live without larger populations of seals and, ultimately, phytoplankton, so, this logic says, a large IT vendor needs a population of small complementers, such as value added resellers, application developers and service providers in order to strive and grow. But, unlike in ecology, it is not straightforward to identify any hard law (such as the conservation of energy) which justifies this. The small business partners can be needed for many things - new customer contacts, local support, new ideas of applying technology to specific domains, or feedback on desired features and usability. Porter's five forces analysis ([Por79], [Por80]) suggests that it is better for the big vendor to have many small partners instead of a few large ones, as he can then yield greater negotiating power. This general rule has turned out to be wrong in some important cases, as shown later.

3 Case studies of software platform ecosystems

The Digital Business Ecosystem project is somewhat peculiar because its project plan largely parallelizes requirements analysis, design and development ("computing"), and implementation and evaluation (socioeconomic analysis), activities which are done in sequence in conventional projects. The intention of work package 13 on business networks is to propose design criteria for ecosystem platforms based on economic modeling and on empirical data. For the empirical part, the DBE cannot be studied, since the DBE infrastructure is only being developed and not used in real-world cases. Small, localized platform projects are not too well-suited either, because one cannot be sure that the actors in these cases thought carefully about their actions, and for some of them their participation in the ecosystem may only be marginal compared to their main lines of business. Very recent projects are also not well-suited, because success or failure can often only be evaluated with several years of distance. Therefore, we have looked for major platform cases, where several million or even billion dollars were at stake for the participating companies, and whose history can be traced back 10 years or more. Two of these cases are presented in this report: Video consoles, and Lotus Notes.

3.1 Video consoles

The market for home video consoles and games, although quite far removed from the B2B problems addressed by the DBE project, is of special interest for several reasons: It represents in many ways a "minimal" configuration of an ecosystem; for the longest time of its history, it was characterized by only two types of players: console vendors and game publishers. It is a fairly closed system: There is only some limited interaction with the general computing equipment industry, and with the motion pic-

ture industry for games based on their themes. Its history goes back almost thirty years, with seven fairly clearly identifiable generations of console hardware.

3.1.1 Timeline

[2] Home video games started in 1972 with Magnavox Odyssey and became popular in 1975 with a home version of the popular arcade game *Pong* by Atari. These very first video games integrated hardware and software, and so each console only ran one game, which consumers soon found boring. In August 1976, Fairchild Semiconductor introduced the VES / Channel F system with replaceable cartridges. This quickly became the dominant design of the industry: Atari and other vendors quickly followed with similar systems, which together are commonly called the **first generation** of home video game systems.

Although using replaceable cartridges, the business model of these vendors still consisted of making both the console and all the games. This allowed them to subsidize the console by the large margins earned on the games. Some disgruntled Atari developers realized this as an opportunity and started Activision, the first pure game software company. Atari tried, but had no legal means of preventing other companies from selling compatible cartridges for their console. The availability of games from other publishers added to the appeal of the console, but also cut into the console vendor's profit margins.

The **second generation** of consoles was started in 1980 by Mattel. This console was clearly superior in graphics and sound, but was not able to gain much market share because Atari was already perceived as a "standard" with a greater variety of games, and because it lacked a big arcade hit game that customers would recognize. Atari followed up as did two new entrants. None of Atari's competitors from the first generation made it to the second. Independent game publishers became even stronger during this time, and even Atari's second-generation console competitors made successful games for Atari's consoles. The second generation finally ended in the 1983 video game crash, caused by high console inventories, an abundance of low-quality games and a price war for home computers, which became a similarly-priced substitute.

The **third generation** started in 1985 with the introduction of the Nintendo Entertainment System (NES), still based on 8-bit technology. Nintendo's business model showed that it had learned the lessons from the previous generations: Independent game publishers were part of this model right from the beginning, but Nintendo kept tight control of the game supply buy requiring game publishers to sell the games on Nintendo-produced

[2] This section is based on [GP02], [Con], and Wikipedia: "Computer and video game industry", "History of computer and video games", and related entries.

cartridges, and by using a hardware protection mechanism which locked out anybody who tried to clone them. In addition, Nintendo developed quite a few successful games in-house. These policies would eventually lead to an anti-trust enquiry but gave Nintendo enough time to establish itself as the leading vendor.

The **fourth generation** was marked by the successful entry of Sega in 1989 and the transition to 16-bit technology, but the business models remained largely the same. Due to the increasing sound and graphics abilities, game development became more and more challenging. While during the 8-bit era, individual programmers sometimes almost single-handedly created blockbuster games, doing all the programming, graphics, sound and instructions themselves, game development became more professional, with individual specialists, although still inside one team. This was accelerated by the introduction of CDs which were able to store much more graphics and animation sequences.

The **fifth generation** saw the introduction of 32-bit systems by Sega (Saturn) and Sony (Playstation) in 1995, able to produce more realistic 3D graphics. Sega introduced an online service for the Saturn, although this was not perceived as a killer feature at the time, and could only be used with an add-on. In 1993, Atari had already made a last effort in the console business by introducing Jaguar, usually considered to be a fifth-generation console. It failed because it was difficult to program and had some hardware bugs, so few games were published for it. 3DO, a startup company, had generated significant interest in 1994 by introducing a 32-bit gaming technology licensed to several hardware vendors and game software companies. This led to high console prices, however, as there was no incentive for hardware vendors to subsidize their production, and the concept soon failed in spite of broad support by game publishers.

Sega started the **sixth generation** in 1999 with its "128-bit" Dreamcast system, also capable of online gameplay. Sony responded over a year later with the Playstation 2. But two strengths made the latter very successful: The ability to act as a DVD player, and its backward compatibility with the previous Playstation. In 2001, Nintendo's Gamecube was introduced and Microsoft entered the market with the Xbox. Independent game publishers increasingly made their games available across several platforms, including the PC. This may also be the result of the ever-increasing proportion of recorded graphics and sound content, which can be quite easily transferred across platforms, compared to programmed content. All console vendors now offered online gameplay, although in different ways. Microsoft and Sega host a central service where all games are played, including those of independent game publishers. Sony's approach is more decentralized. Users have to pay a monthly fee for most services.

Microsoft intended to attract game developers by the similarity of the Xbox to the PC platform, and thus very powerful existing tools for software

development. This was one phenomenon of a larger movement which has grown even stronger in the following generation: The increasing modularity of game software, the use of standard APIs for specific purposes across consoles, and the emergence of game software component vendors ("game middleware").

The **seventh generation** has just started at the time of this writing with the release of the Xbox360. The Playstation 3 and Nintendo Revolution are scheduled to appear in 2006.

3.1.2 Persistent success factors

[GP02] focus on persistent success factors typical for network industries in their detailed analysis of the video game industry. They list Dominant Designs, Switching Costs vs. Technological Innovation, Entry Timing (First Mover Advantages), Complementary Products & Installed Base, and Tipping (Market Share) as potential factors or phenomena determining the success of a company in such industries, in addition to the factors well-known from other sectors. They conclude that dominant designs, switching costs and tipping were not decisive factors in this case. Also, while the installed base and complementary products are very important to establish leadership during one generation, they do not deliver very strong advantages for the following generation. It is striking how often market leaders became almost insignificant within a few years, and how new entrants could achieve significant market share within one generation. According to [GP02], there were few systems which offered backward compatibility, especially in the early generations. One reason for this may be the fact that these games were mostly played by children, who play these games for a few years and then lose interest in video games. Therefore, there is little incentive for vendors to compromise on the design or on production costs in order to maintain backward compatibility. This typical customer profile has certainly changed in recent years, with more and more players in their thirties, and much attention given to "hardcore gamers" who are willing to pay expensive monthly subscription fees.

We see another possible explanation for the fairly low entry barriers in this market: The fast rate of technological progress of the underlying computer technology, and their high added value from the consumer's point of view. The technological progress is largely exogenous and therefore cannot be influenced strategically by any vendor, but the value consumers attribute to it depends on new game ideas which make use of the added computing power. While the past few years were very much centered on realistic 3D graphics performance, it could well be that in the near future, consumers will tire of this aspect. Some industry analysts have already made out a "creativity crisis", with many current games being only graphically updated versions of old games.

Until very recently, accessing the added power required the use of new programming interfaces (APIs). Therefore, using an old game software on a new backward-compatible cartridge would not significantly improve performance, and lead to a far less satisfying experience than a new game. Therefore, even if the new cartridge was not backward compatible, consumers would not undergo any switching costs (they could still turn on their old console for playing old games). This is very much different from enterprise software, where the software customer can only effectively run one version of the software at any given time, and it is vitally important to transfer all the old data when switching platforms.

First mover advantages and technological innovation, along with complementary products for the current generation, most clearly determine vendor success according to this analysis.

Another important lesson to be learnt from the Atari case is that platform success does not always equal vendor success. A vendor has to walk a fine line between being open enough to be attractive to complementers, while maintaining sufficient control of the revenue streams around the platform.

3.1.3 The changing network

While [GP02] were more concerned with the persistent success factors, we focus on the changes in the video game ecosystem since its beginning. The following figure shows how this ecosystem has become more complex over time, with more and more specialized roles. The diagrams reduce the actual complexity in each configuration somewhat, but still illustrate the major changes.

The main changes since 1999 have been the increasing importance of online gaming, and the emergence of modularity, standard APIs and game middleware. Modularity became a visible phenomenon with so-called first-person shooter (FPS) games. The most impressive feature of this genre at the time was the 3D realism, implemented by a 3D engine which was able to use the latest 3D accelerating graphics cards. These engines required a considerable effort of engineering, and were consequently reused in several games. *id software*, the studio creating the most successful titles in this genre, historically had used shareware as a marketing tool and, therefore, had a liberal policy of sharing the engine binary, and eventual its source code, with customers and other game studios. Not only did *id* separate the 3D engine from the other game content, but they also separated the engine from the system platform in a certain way. For example, *id*'s 3D engine uses the OpenGL standard for 3D rendering wherever possible to access the graphics hardware instead of Direct3D, which is limited to Windows, and actively engages with graphics card vendors to improve the standard. Also, because it was originally developed for PCs, the engine adapts auto-

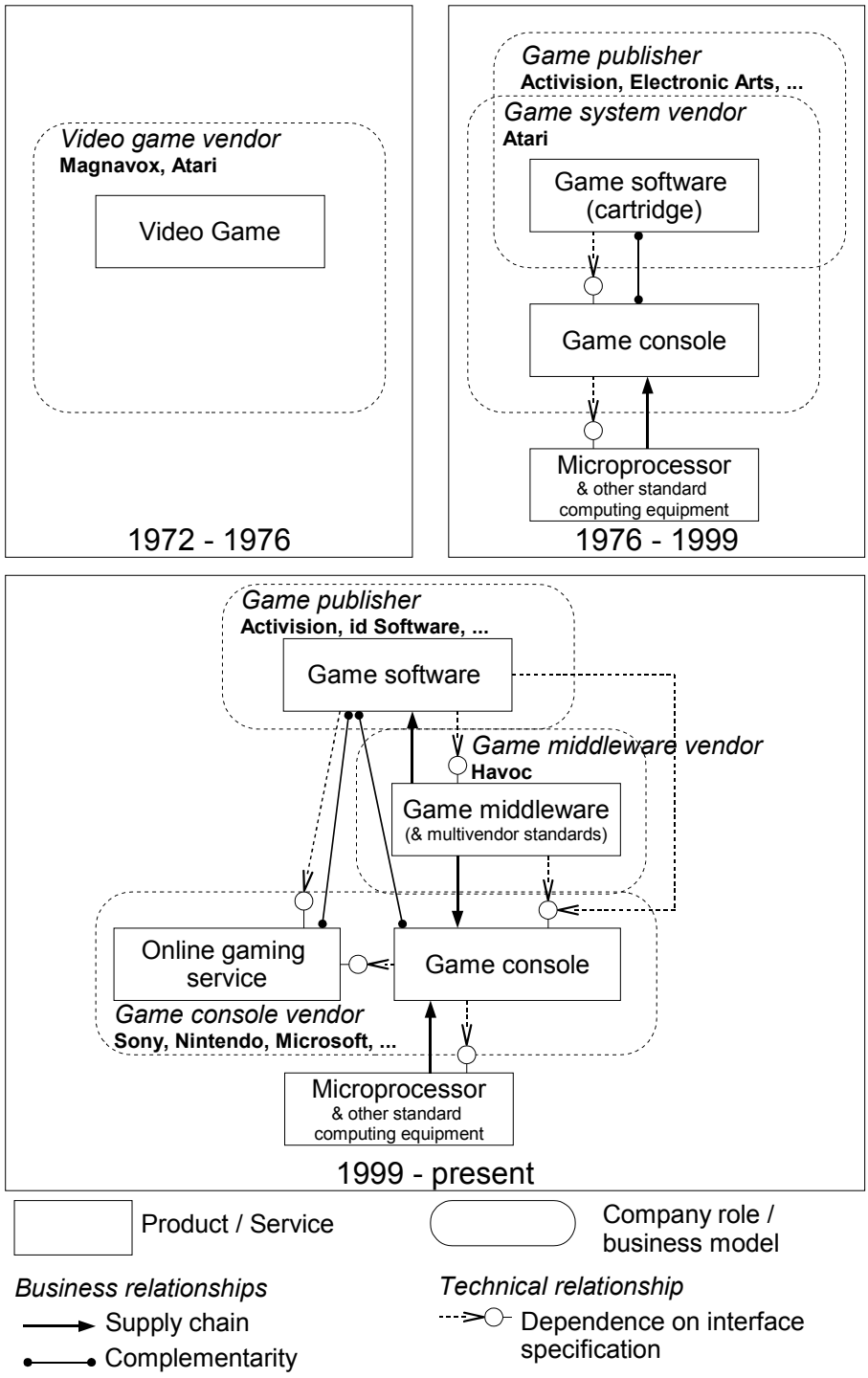

Figure 1: Evolution of the video gaming ecosystem

matically to the amount of computing power and screen resolution available, so that rendering is smoother on better hardware. This is different from the video console games up to the fifth generation, where games had to be rewritten for every hardware generation in order to take advantage of it. These measures have made porting the engine and games to several video console platforms fairly easy.

One of the most popular games for online network play in recent years, Valve Software's *Counter Strike*, illustrates the potential of this modularity. Valve Software had licensed the source code of the Quake engine (a FPS game by *id*), modified it, and published their own game *Half Life* based on it, with considerable success. *Counter Strike* was a modification carried out by a Vietnamese-Canadian college student using the tools provided by Valve. Valve then bought the publishing rights to this modification.

Another recent phenomenon is the emergence of specialized game component software vendors. For example, the Irish Havok.com Inc. specializes in providing so-called physics engine, for realistic simulation of game mechanics, such as collisions, explosions or water waves. Havoc delivers both to game studios who include the engine in their games and to 7th generation console makers who include it as part of the "operating system", thereby making the console more attractive to game developers.

The list of technologies to be included in Sony's Playstation 3 underlines the importance of middleware and standard APIs[3]:

Open standards:

- COLLADA, an open, XML-based file format for 3D models;
- OpenGL ES 2.0, the embedded version of the popular OpenGL graphics API;
- OpenMAX, a collection of fast, cross-platform tools for general "media acceleration," such as matrix calculations;
- OpenVG, for hardware-accelerated 2D vector graphics.

Sublicensed technologies:

- Ageia's PhysX SDK, NovodeX (physics engine);
- Epic's Unreal engine 3.0 framework (first-person shooter engine);
- Havok's physics and animation engines;
- Pixelux's Game Asset Synthesis Technology, a toolkit for advanced procedural synthesis;
- Alias Systems Corporation's 3D graphics programs;

[3] from Wikipedia "Playstation 3"

- Cg, Nvidia's C-like shading language;
- SpeedTree RT, a programming package produced by Interactive Data Visualization, Inc. that aims to produce high-quality virtual foliage in real time.

Finally, online gameplay is becoming more and more important. This could be a very important development. Up to the fifth generation, as long as there was an effective copy protection mechanism in place, network externalities related to the number of players were limited, unlike in the PC industry, where people exchange software, e-mails, documents etc. (There were *indirect* network externalities based on the interaction between software publishers and number of customers - customers buy consoles with many available game titles, game developers are attracted to consoles with many customers.) With online gameplay, customers have a direct added value from using a popular console, and, even more clearly, a popular game. Already, game publishing is a risky business due to *attention economy* effects, and a few top games usually capture half of the sales volume, while many titles sell only a few thousand copies. With network effects, this phenomenon can be expected to grow stronger, and the high risk associated with game publishing will likely lead to a further concentration of a few big publishers, along with small studios which come and go, similar to the pharmaceutical industry which also has this risk and sales structure. The longer customer life cycle together with the direct network effects in relation to individual games and the availability of cross-console middleware could over time give more power to the big game publishers relative to the console vendors.

3.2 Lotus Notes

Lotus Notes/Domino by IBM is one of the leading collaboration suites today (it was called a "groupware" through much of its history, which means the same thing). It offers such functions as e-mail, form-based data entry, to-do lists and instant messaging. Its overall success is mixed: While it is certainly, together with Microsoft Exchange, the dominant collaboration suite in the industry, and especially well-established in larger corporations, many analysts would agree that its potential as a concept were even greater, notably as a platform for mission-critical applications such as CRM and for data-oriented web solutions. CRM (Customer Relationship Management) accounted for a considerable part of the total enterprise software market in recent years, but is completely dominated by vendors like Siebel and SAP who use standard relational databases as their platform. This is surprising as Lotus Notes had many of the typical CRM functions implemented many years before these vendors, and the first such systems had been developed

on top of Notes by Lotus' business partners. In the area of data-oriented web solutions, technologies based on the Apache webserver and Microsoft IIS are dominant, with Domino mainly used for internal applications.

We think that Lotus failed to establish itself in these more mission-critical, and more lucrative areas because of the lack of a standard integration architecture for their many business partners and complementers, and that Lotus scattered their efforts on many related areas instead of focusing on making the core technology competitive with the alternatives in each area. We will therefore look especially at efforts to establish such an architecture, the need for which was undisputed, by different stakeholders and the areas in which Lotus engaged at different times.

The origins of Lotus Notes go back to the PLATO project at the University of Illinois. In 1973, PLATO Notes was released, mainly as a secure bug tracking application ([Lot03]). In 1976, a much more powerful PLATO Group Notes was released, which already contained many of the functions defining the groupware genre, and even a platform for multiplayer games. This system was very popular and well-known within the world of Mainframe computers.

In 1984, Ray Ozzie, who had worked as a programmer on the PLATO system, founded Iris Associates with the help of Lotus software, in order to work on a PC version of Notes, including some novel ideas of his own. Around 1988, even before the first release, Lotus (who acted as a distributor) was able to sell 10,000 licenses to Price Waterhouse Coopers, as extraordinary deal at the time. Release 1.0 was shipped in 1989.

Lotus did see a future potential in small and medium businesses, but thought that only large companies were ready for the new concept. Therefore, they saw their role as interacting directly with all clients; partners (value added resellers, solution developers) were not part of the picture ([Kee90]). In 1990, Lotus founded the Consulting Services Group, which would do the necessary customization ([Bar92]). The technology itself, however, was always geared towards custom development of domain-specific applications. Release 2.0 (1991) included an API for the C programming language, which allowed the development of professional plug-ins.

Around the introduction of Release 3.0 (1993), Lotus started targeting the larger market, and many small business partners established themselves, offering custom solution development. Beside North America, Notes became very popular especially in Germany, where it was introduced by large chemical companies (BASF, Henkel, Bayer) who saw its potential([Weg05]). In these larger installations, the lack of a *domain architecture* on top of the technological platform soon became apparent. Lotus Notes as a product delivered only very basic predefined dataset definitions, but even the predefined address book was far from what an organization needs in reality. A domain architecture organizes the domain objects (such as contacts, employees, projects etc.) into several modules or layers (such

as an industry-independent layer and several industry-specific modules), and is used by almost every software company that needs to efficiently cater to different types of customers. The technology platform of Notes mainly defined technical objects such as fields, data types, or records.

Therefore, every solution developer working for several large customers developed their own domain architecture and industry-independent application core. This would allow modules that had originally been developed for different types of clients and by different teams to be integrated somewhat smoothly, but only within one solution provider. It would in general not be possible to combine modules developed by different solution providers in any meaningful way, since each module depended on their version of an address book, organization model etc. It also raised switching costs for end customers for changing solution providers. Important solution providers in Germany during this time were Gedys, Group Technologies, Intraware and Teamwork.

Many solution providers, however, saw this not as an advantage (being able to lock in their customers), but as an obstacle to entering more serious, mission-critical areas. A few of them made efforts for standardardization of an application core, both by raising the issue with Lotus and by publishing proposal themselves. However, this was not met by much enthusiasm by Lotus in North America, where there were perhaps fewer efforts in using Notes for heavier applications.

In 1995, Lotus was purchased by IBM ([Sch95]). This helped Lotus achieve more credibility with large accounts and gave Lotus access to a large global sales force. Version 4.0 was released in 1996, with Domino, a former web server product by IBM integrated, making it one of earlier products that had successfully made the transition from the Client/Server to the Web world.

IBM Lotus began selling Consulting Services more aggressively. At first, the idea was to concentrate on the very big accounts, leaving the small and medium-sized customers to the partner network. But around 1997, at least in Europe, Lotus Consulting also went after medium-sized customers, perhaps because many of the large companies already had an established solution provider. This created considerable tensions with the partner network. Lotus Consulting soon ran into the problems with domain architectures themselves in their projects and started promoting their own application core, called LSA (Lotus solution architecture) around the same time. This initiative was welcomed in principle by large customers and even solution partners, but its implementation brought tensions with solution partners to a boil around 1998 ([Som98]. For established solution partners, LSA would invalidate investments made into their own domain architectures and create a lot of effort for migrating existing applications which many of their customers were not willing to pay for ([CWS98]. In addition, Lotus Consulting wanted to charge hefty per-seat licenses for LSA. As not

all existing customers would be willing to pay for LSA, solution providers would have had to maintain two versions of their solutions in parallel - one stand-alone, one based on LSA. Also, LSA not only contained the minimal set of cross-industry, cross-functional domain objects, but many functionalities which were the bread-and-butter business of solution partners. LSA was also criticized for requiring the installation of special DLLs, compromising the cross-platform capability of Notes. For small and medium businesses, LSA was simply too expensive and complex, and many demanded that a basic LSA be built into the standard version of Notes.

As a result, and because, as a German/EMEA initiative it did not receive backing from Lotus in the U.S., LSA was slow to progress. In the meantime, a Danish startup company, IT Factory, had appeared, proposing their own *(ITF architecture)* as a standard application core. Initially, Lotus Consulting EMEA and IT Factory worked together towards integrating both their efforts. At the beginning of 1999, IT Factory announced that Lotus had agreed to transfer LSA to IT Factory ([Som99]. Lotus never confirmed this, and so relations turned very sour between the two companies ([Web00]). IT Factory continued to aggressively market their initiative, even distributing it for free to other solution developers for a few months ([Hei99]. In 2000, they offered a unified framework for both Notes and Microsoft Exchange ([CWI00]) which enabled a certain degree of interoperability between the two platforms. But then, in 2001, IT Factory collapsed like so many IT companies during that year due to its exaggerated, venture-capital-driven expansion plans ([Web01]). LSA was revived as LiSA (Lotus i-Net solution architecture), but still received criticism from solution developers ([CWK01]).

Meanwhile, other, more specialized platforms appeared and established themselves in the areas of CRM and web application, closing the window of opportunity for Lotus. Notes/Domino did continue to be successful as a suite for standard collaboration tasks, and has been integrated with IBM's Websphere offerings in recent years. However, the early announcement of the "Workplace" strategy created insecurity with potential Domino customers. This, together with the shortcomings and complexities of Notes as a development platform, with its mixed macro and script languages, has intensified the shakeout of Lotus solution partners after 2000 ([CWK01]). The new "Workplace" platform faces similar problems in that it integrates a little bit of many platform categories at the same time, such as web development, messaging, complex application development, form-based data entry, but in each of these categories there are better, more focused and less complex solutions ([CZW05]). On the other hand, there is almost no alternative delivering this breadth of built-in functionality, and future success will depend on how many applications can be identified which can benefit from the high degree of integration provided by the Workplace suite.

4 A qualitative model of platform ecosystem evolution

The cases presented show how the evolution of technology platform ecosystems is influenced by three forces, in addition to the factors found in other industries:

- The base technology on which the platform is implemented;
- the alternative, competing platforms;
- the network of users and complementers of the platform specification, with the potential emergence of middleware and, ultimately, higher-level platforms.

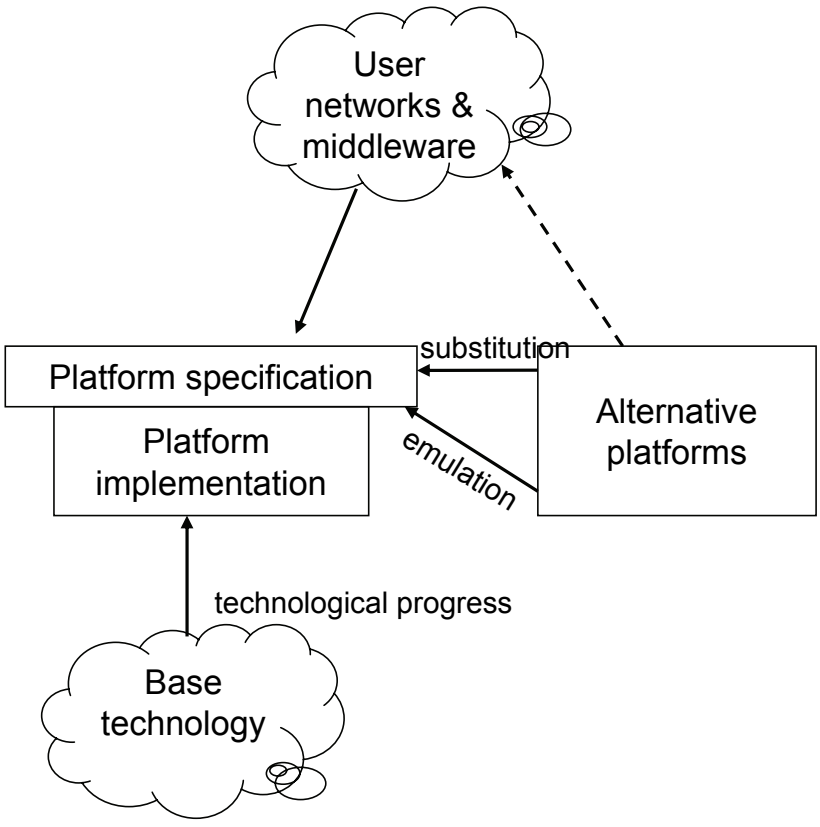

Figure 2: Three forces acting on a platform

The characteristics of the competition in the ecosystem change over time, depending on the life-cycle phase of the ecosystem. We can roughly

identify four phases:

1. *Era of ferment.* During this early phase, competition is fairly weak, as the main barrier to overcome is awareness of the technology by its potential users. Technology vendors largely have a common interest in generating attention. There is no common understanding yet in how to implement the technology, and the boundaries of the market are not clearly defined.

2. *Win or Die.* This is the early phase after the establishment of a *dominant design*. Customers and potential complementers have now understood the platform concept and compare platforms based on features, performance and credible backing by a large investor, vendor or customer. It is clear that before long, only a fraction of the current platform vendors will be able to survive, due to the strong network effects in the interaction between complementers and end customers.

3. *Build friendly user network.* After the initial shakeout, the remaining platforms will become more attractive and more secure to complementers and customers. They can now afford to make significant long-term investments into the platform. This is good for the platform in general, but contains the danger of a "Java attack" by a middleware vendor.

4. *Performance* Eventually the platform will lose its appeal to the large networks of application developers because better, more productive platforms are available on top of it. Then the implementations of the platform will be measured by their performance characteristics alone.

In the era of ferment, finding the right combination of features and price/performance point to make inroads into mainstream customers is critical. As a first mover, one can perhaps secure a lasting advantage for the next phase (depending on the strength of direct network effects).

In the *Win or Die* phase, the goal for a platform vendor must be to drive out as many competitors as possible completely, in order to stay in the game himself. When a vendor already has a strong position, he should usually not make his platform interoperable with others. However, when there are a few contenders of similar strength, users or complementers have to decide between platforms, and the utility of the platform is essentially linked to network effects, this can lead to a deadlock situation, where creating some limited interoperability with the competition might be the only solution, even though one should retain some form of competitive advantage. This is the current situation of the HD-DVD vs. BluRay battle, where both lose if there is not a decision soon.

The establishment of a dominant design for a new platform may be the end of a network formerly linked to the base technology. This base technology will now be measured by its performance only.

In the *Network* phase, a few platforms are usually firmly entrenched in their respective communities, and it has become unfeasible and too costly to continue the platform battle with full energy. Instead, if barriers between platforms are maintained, there is the danger of creating a great business case for cross-platform middleware in various application areas, which would undermine the negotiating power and sales relationship of the platform vendor to his customers. It could therefore make more sense to establish interoperability with other platforms to some degree. This seems to be currently the case with .NET and Java. Both Microsoft and Sun know by now that they will not be able to kill the other technology, so they have little to lose and much to gain from better interoperability. This way, it is easier to keep complementers happy who are specialized in one platform. Java is itself an example of a middleware whose creation was spurred by the gulf between Microsoft Windows and various flavors of Unix. In addition, it addressed not only a specific application area of the original platforms, but provided a general-purpose API, threatening to potentially dissolve the entire network connected to the original platforms, therefore it is appropriate to qualify it as an attack on Windows itself. (Fortunately for Microsoft, the implementation, initially slow, took some time to catch up with the initial vision.)

Whether or not a platform ever reaches the *Performance* phase depends on several factors. First of all, there can be a significant progress or change in the base technology which makes some essential assumptions in the design of the platform obsolete. In case of such a disruption, the entire technology stack has to be recreated layer by layer. Such revolutions were common early in computing history, e.g. with the transition from batch to online systems, or from text-based to graphical terminals. They have also occurred until very recently with every single generation of video game consoles, and they can still be observed in mobile computing.

While it is impossible for a platform to deal with fundamental revolutions in the underlying technology, some platforms have been constructed to scale with its linear improvement. Probably the first such platform in the hardware area was the IBM System/360 introduced in 1964, which would scale from the then low-end to high-end systems. Early PC manufacturers, including Apple and IBM itself, were not able to repeat this, and their machines threatened to become out of date after a few years. [GC02] describe in detail how Intel realized this problem, which hampered sales of their more advanced chips of the x86 family, and how they put in a lot of effort with other manufacturers in the early 1990s to define the scalable PCI bus standard.

If the platform is scalable, then the *Performance* phase is reached once the majority of the application development takes place on higher levels of the technology stack. The former platform product has then itself become a base technology for these higher level platforms, and it will be measured mostly by its performance/price ratio. One such example is the Intel x86 architecture just mentioned, used for servers. Most application development on the server today takes place using higher level programming languages and portable APIs. Therefore, while Intel could afford to have inferior performance/price with respect to other architectures a decade ago, when the computing world was still divided between processor architectures, today they are successful precisely because they can compete on absolute performance. The same even applies to some extent to server operating systems such as Windows 2003. There is a lot of attention on performance benchmarks for database servers, file servers, e-mail servers etc. - these have become the real application platforms. However, maintainability features of the operating system also count.

The transition from the Win32-API to .NET can also be explained by this phase. Win32 is the common API introduced for application development across the Windows 95 and NT product lines. Because it is based on an old, imperative style of programming, most developers, by the year 2000, had switched to tools and APIs on top of it, hiding the ugly details from them. Microsoft themselves offered Visual Basic Components (VBX) and Microsoft Foundation Classes (for C++), but developers had begun moving towards Java and Web frameworks. A new API was needed to keep the relationship with application developers, and .NET succeeded by emulating the perceived advantages of Java and adding a very productive environment. But underneath, the Win32 is still being used internally, since it scales well with the advance in the underlying technology. It is merely a vehicle for the new .NET API, and can be replaced once the existing applications have been migrated, and if there is somehow a better vehicle for .NET.

5 Game-theoretical analysis of factors influencing the licensing game

In the last report *Economic models of software ecosystems*, a certain class of games was presented which seeks to model typical strategic interaction among complementers for a given platform. Vendors engage in a game with two steps; in the first, non-cooperative step, they decide on their development plan, i.e. which components and products to develop. In the second, cooperative step, they negotiate licensing components, and prices for these licenses. Formally, we determine a reasonable payoff from the second step using a modified *fair allocation rule*, and then use them as payoffs

for the first stage game in order to identify Nash equilibria. (The computational complexity only allows us to find the pure Nash equilibria so far). Using this model, we hoped to investigate factors, such as characteristics of the technology or domain structure, the legal framework and communication channels, fostering or hindering the establishment of software ecosystems based on a common technology platform. More precisely, we were interested in the influence of the legal framework, changes in overall productivity, and cost asymmetries between vendors. This section presents some results of this investigation.

The notation used is the same as in the previous report: Vendors are numbered 1,2,3,...; products denoted a,b,c,...; components z,y,x,.... r is the revenue function, $v_S(C)$ is the modified characteristic function, i.e. the joint payoff that coalition S can achieve under cooperation structure C. $\psi_i(v, C)$ is the post-transfer payoff vendor i is predicted to achieve in cooperation structure C according to the modified fair allocation rule:

$$\sum_{i \in S} \psi_i(v, C) = v_S(C), \forall C \subseteq L_2(N), S \in N/C; \tag{1}$$

$$\psi_i(v, C) - \psi_i\left(v, C \setminus \{\{i, j\}\}\right) = \psi_j(v, C) - \psi_j\left(v, C \setminus \{\{i, j\}\}\right), \quad \forall C \subseteq L_2(N), \{i, j\} \in C. \tag{2}$$

Let $w_i = \psi_i(v, L_2(N))$ denote the payoff player i can expect under full cooperation, which is the payoff used for the first-stage, non-cooperative game.

5.1 Global cost/revenue ratio

First, consider the technology network presented in the last report, a fairly "dense" technology network:

n_a	r_a
1	100
2	40
3	10

		n_b			
r_b, r_c		0	1	2	3
n_c	0	0,0	80,0	30,0	0,0
	1	0,60	50,40	10,10	0,0
	2	0,20	20,10	0,0	0,0
	3	0,0	0,0	0,0	0,0

Table 1: Low revenue function

In the symmetrical configuration presented above (all vendors have the same development cost), and under the assumptions of the extent of contracts made in the last report, there were the following two Nash Equilibria (except for permutations):

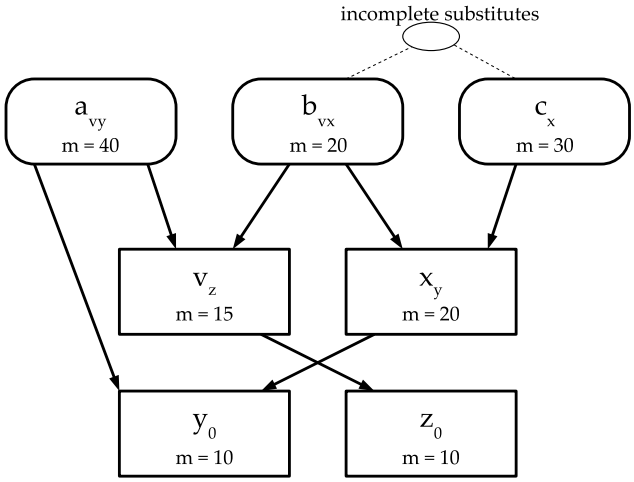

Figure 3: Example technology structure

1. One vendor produces everything except c_x, the others nothing. He receives 65, the others nothing.
2. Vendor 1 produces a_{vy} and all the necessary components v_z, y_0, and z_0, and receives 25. Vendor 2 produces b_{vx} and all the necessary 4 components, and receives 5. Vendor 3 produces and receives nothing.

What happens if the general ratio between development costs and revenues changes? Will a more lucrative market support more vendors and more collaboration, or will there be the same equilibria with proportionally higher payoffs?

This question was studied based on the same game, with a higher revenue function, and varying cost scales. This is the game with "normal" costs:

n_a	r_a
1	180
2	80
3	20

		n_b			
r_b, r_c		0	1	2	3
n_c	0	0,0	200,0	90,0	0,0
	1	0,130	130,70	40,30	0,0
	2	0,50	80,20	10,0	0,0
	3	0,0	30,0	0,0	0,0

Table 2: High revenue function

A note about how revenue functions were constructed: The basic principle of increased competition can be seen with the r_a: The total revenue decreases as more vendors offer the product (from 180 to 160 to 60). If products were completely equivalent, then as soon as there are two vendors or more, there would already be Betrand price competition, driving

down revenues to marginal cost. (Price competition is appropriate because of the absence of capacity restraints. Marginal costs are near zero.) However, precisely because of this, and because of their complexity, software products and vendors' distribution channels are never completely equivalent. We can imagine that, as the number of vendors increases, a modest increase in units sold partially compensates the drop in prices.

B and C are partial substitutes; in this case, more customers prefer B. Revenues for B drop as the number of vendors for C increases, but not as much as if these vendors offered B as well.

This game leads to the following pure Nash equilibria:

	q_1	q_2	q_3	w_1	w_2	w_3	$\sum w$
1.	ac-vxyz	ab-vxyz	b-	55	35	10	100
2.	c-vxyz	ab-vxyz	ab-	$61\frac{2}{3}$	$41\frac{2}{3}$	$16\frac{2}{3}$	120
3.	ab-vxyz	ab-vxyz	-	55	55	0	110
4.	ab-vxyz	b-vxyz	a-	$61\frac{2}{3}$	$61\frac{2}{3}$	$6\frac{2}{3}$	130

Table 3: Nash Equilibria for standard symmetric game

(All permutations of these equilibria are also equilibria, as all vendors are equivalent.) What is interesting in equilibria 1,2 and 4 is that some vendors produce components which they cannot use themselves (e.g. c-vxyz). In fact, they use them as a lever in negotiations with the other vendor who makes his own components. In these equilibria, there is always a complementary vendor who does not have the necessary components for his products, which are more in competition with the third vendor (e.g. ab- with ab-vxyz) than with the one who is making the "unnecessary" components (e.g. c-vxyz). c-vxyz has higher negotiating power because he can threaten ab-vxyz with giving components to ab-. At the same time, under the full cooperation, this is ultimately avoided, so the total payoffs are higher (as there are fewer vendors) than if c-vxyz actually started selling other products himself.

We will call such equilibria *"threatening-to-license"* equilibria, as opposed to *autonomous* equilibria such as 2. where in each player's development plan, all necessary subcomponents are developed for each product developed, and so licensing is not intended nor does it affect the payoff distribution. Finally, we can imagine *licensing* equilibria where components actually are licensed between players, as well as hybrid equilibria where several of these phenomena appear.

If all costs go up by a factor of 1.1, there are only three equilibria:

Equilibria 2 and 3 correspond to equilibria 3 and 4 in the standard case. Equilibrium 1 is a threatening-to-license equilibrium, where vendor 2 would need *vyz* from vendor 3 in order to produce *b*.

With a factor of 1.3, only the autonomous equilibrium 2 (in table 4) is

	q_1	q_2	q_3	w_1	w_2	w_3	$\sum w$
1.	ab-vxyz	b-x	a-vyz	$49\frac{2}{3}$	$7\frac{2}{3}$	$48\frac{2}{3}$	106
2.	ab-vxyz	ab-vxyz	-	43	43	0	86
3.	ab-vxyz	b-vxyz	a-	$49\frac{2}{3}$	$7\frac{2}{3}$	$48\frac{2}{3}$	106

Table 4: Equilibria for game with costs scaled up by 1.1

possible.

With a factor of 1.5:

	q_1	q_2	q_3	w_1	w_2	w_3	$\sum w$
1.	ab-vxyz	-	-	207	0	0	207
2.	ab-vyz	b-x	a-vxyz	$3\frac{2}{3}$	$6\frac{2}{3}$	$3\frac{2}{3}$	14

Table 5: Equilibria for game with costs scaled up by 1.5

Equilibrium 1 is a new autonomous equilibrium which corresponds to the first equilibrium in the low revenue case (table 1). Equilibrium 2 is an interesting hybrid of a licensing and a threating-to-license equilibrium. Without any licensing, vendor 1 will only be able to offer a, and vendor 2 will not be able to offer b. Vendor 3 develops an x that he does not need for a. But when vendors 1 and 3 cooperate, vendor 3 can license this x to vendor 1 who can then offer b (the competition between the two vendors regarding a is not affected by this license, a formal example of *co-opetition*. But vendor 3 can also license x to vendor 2 who can then offer b. Vendor 3 could also license x to both other vendors. But this would be bad for the total revenue, so the negotiations in this case lead to an exclusive license of x to either one of the other vendors.

For factors 1.6, 1.7 and 2.0, only the first equilibrium in table 5 is preserved.

For a factor of 0.5, only equilibrium 3 in 3 is possible.

We can conclude from this example that the global cost/revenue level *has* a strong influence on the viable equilibria. When costs are low and revenues high, more vendors seem to be able to survive with autonomous strategies. When costs are high and revenues low, one vendor is selected. Threatening-to-license equilibria appear mostly in between the extremes, and are more common than licensing equilibria.

5.2 Technology structure

The fact that there are few licensing equilibria may not be very satisfying in a study about cooperation between vendors. Perhaps this has something to do with the quite "dense" technology structure, where most of the components are necessary for any product, and so it is hard to divide the market?

Consider the following modified technology structure; revenues shall be the same as in table 3.

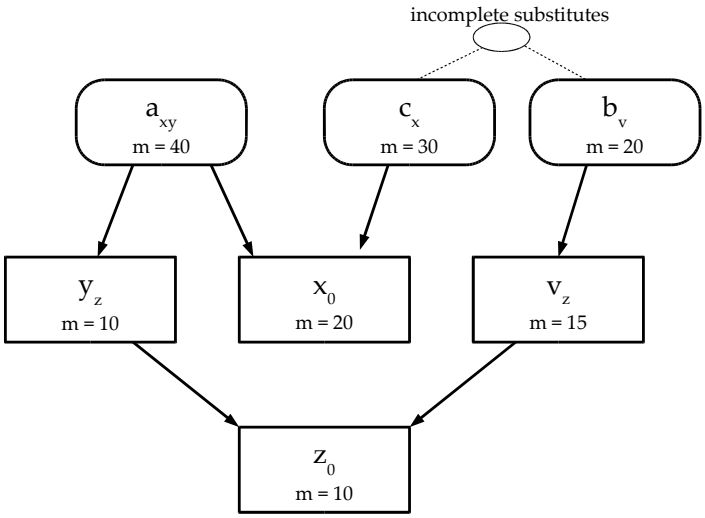

Figure 4: "Sparse" technology structure

The pure Nash equilibria in this case turn out to be (again ignoring permutations):

	q_1	q_2	q_3	w_1	w_2	w_3	$\sum w$
1.	ac-vxyz	ab-vxyz	b-	55	35	10	100
2.	ac-vxyz	ab-	b-vxyz	$61\frac{2}{3}$	$16\frac{2}{3}$	$41\frac{2}{3}$	120
3.	c-vxyz	ab-vxyz	ab-	$61\frac{2}{3}$	$41\frac{2}{3}$	$16\frac{2}{3}$	120
4.	ab-vxyz	ab-vxyz	-	55	55	0	110
5.	ab-vxyz	b-vxyz	a-	$61\frac{2}{3}$	$61\frac{2}{3}$	$6\frac{2}{3}$	130
6.	ab-vxyz	b-vz	a-xyz	55	45	0	100

Table 6: Equilibria for game with costs scaled up by 1.5

Equilibrium 1, 2, 3 and 5 are threatening-to-license equilibria, 4 and 6 are autonomous equilibria. Equilibrium 2 is interesting because there are two threats: Vendor 1 can license vz to vendor 2 to enable him to sell b, which is bad for vendor 3 - vendor 1 would never want to enable vendor 2 to sell a. Vendor 3 can license xyz to vendor 2 to enable him to sell a, which is bad for vendor 1. For all vendors together, it is best not to license anything to vendor 2. Without licensing, the pre-transfer payoffs are: $(55, -60, 125)$. The fair transfers are thus $(+6\frac{2}{3}, +76\frac{2}{3}, -83\frac{1}{3})$.

Another possibility would be a "cyclical" technology structure, as follows:

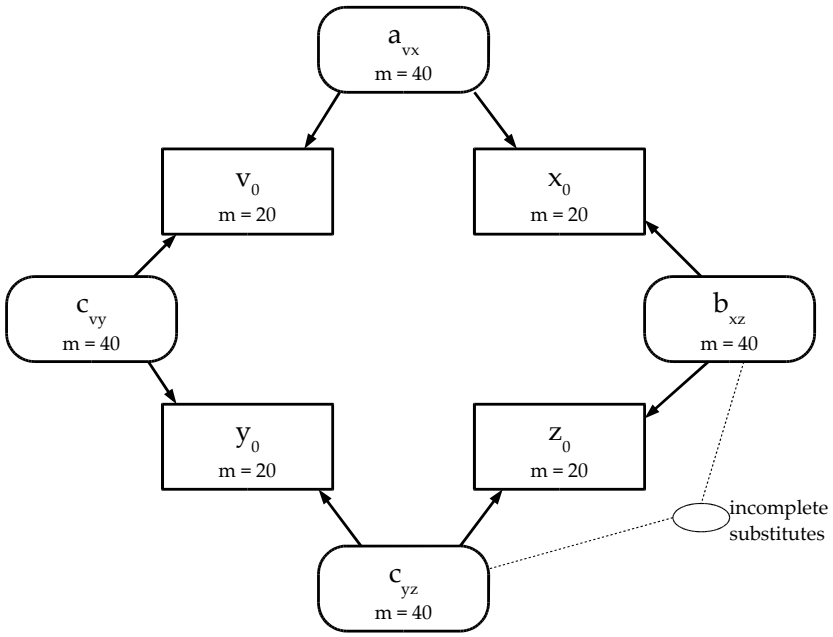

Figure 5: "Cyclical" technology structure

Revenues for *a*,*b* and *c* are as in the standard case, revenues for *d* are similar to *a*'s, but slightly lower:

n_d	r_d
1	160
2	70
3	0

Table 7: Revenue function for cyclical technology

The resulting pure Nash equilibria are:

Equilibria 1, 4, 7, 9, 11 and 12 are autonomous. In equilibrium 1, vendors 1 and 2 both produce *a*, *b* and one of the incomplete substitutes *b* and *c* (each a different one, to avoid direct competition). In equilibrium 9, they both try the individually "best" combination, and the direct competition leads to quite a low total payoff.

Equilibria 2, 5, 6, 10, 13, 14 and 15 are threatening-to-license equilibria.

We can conclude that the technology structure certainly has a strong effect on the viable equilibria, but licensing equilibria are no more common in the alternative structures investigated. Licensing is not unimportant, but it is mostly used as a threat. Perhaps this is not so surprising, given that no vendor has any specific technology advantages. Before looking at cases where vendors have different cost structures, we address another problem

	q_1	q_2	q_3	w_1	w_2	w_3	$\sum w$
1.	acd-vxyz	abd-vxyz	-	40	80	0	120
2.	acd-vxyz	bd-vxyz	a-	$46\frac{2}{3}$	$86\frac{2}{3}$	$6\frac{2}{3}$	140
3.	acd-vxyz	bd-xz	a-vxy	$46\frac{2}{3}$	$51\frac{2}{3}$	$21\frac{2}{3}$	120
4.	acd-vxyz	ad-vxy	b-xz	40	10	50	100
5.	acd-vxyz	d-	ab-vxyz	$46\frac{2}{3}$	$1\frac{2}{3}$	$91\frac{2}{3}$	140
6.	cd-vxyz	abd-vxyz	a-	$46\frac{2}{3}$	$86\frac{2}{3}$	$6\frac{2}{3}$	140
7.	cd-vyz	abd-vxyz	a-vx	20	80	0	100
8.	cd-vyz	ad-vxy	ab-vxz	20	10	70	100
9.	abd-vxyz	abd-vxyz	-	40	40	0	80
10.	abd-vxyz	bd-vxyz	a-	$46\frac{2}{3}$	$46\frac{2}{3}$	$6\frac{2}{3}$	100
11.	abd-vxyz	ad-vxy	c-yz	80	10	10	100
12.	abd-vxyz	ad-vxy	b-xz	40	10	10	60
13.	abd-vxyz	d-	ac-vxyz	$86\frac{2}{3}$	$1\frac{2}{3}$	$51\frac{2}{3}$ 140	
14.	abd-vxyz	d-	ab-vxyz	$46\frac{2}{3}$	$1\frac{2}{3}$	$51\frac{2}{3}$ 100	
15.	bd-xz	ad-vxy	ac-vxyz	$51\frac{2}{3}$	$16\frac{2}{3}$	$51\frac{2}{3}$	120

Table 8: Pure nash equilibria for cyclical technology

discovered earlier.

5.3 Antitrust law

In the previous report, we mentioned the problem that our assumptions then allowed vendors to collaborate even when they could not license any components to each other. This led to phenomena of bribery or cartels in our model, where one vendor would pay another one *not* to license to the third vendor. We can fix this problem by allowing collaboration (more precisely: maximization of joint payoff) only between those vendors who actually have a potential licensing relationship.

Consider the strategy profile (a-xyz, b-vz, ac-yz) in technology structure 4. Vendor 3 lacks x for both a and c. Vendor 1 can license x to vendor 3, which will create competition for his a but is worthwhile since it also enables vendor 3 to offer c, which is only bad for vendor 2. More precisely, while vendor 1 can make 180 in revenues alone (when vendor 3 does not have any revenue), he and vendor 3 together can make 230. For all three vendors, however, the total revenue is 380 when x is not licensed, and 360 when it is. Under the rules used so far, this would have meant that x is not licensed. Vendor 2 would have given a sufficient transfer payment to bribe vendor 1 not to license. According to the fair allocation rule, vendor 2 would have given a transfer payment of $31\frac{2}{3}$ to each of the other players. Under the new rules, vendor 2 is not at the negotiation table, because he has no potential license relationships with the other players. Vendors 1 and

3 split up their gain of 50 evenly, which leads to a transfer payment of 125 from vendor 3 to vendor 1 (the price of the license).

It turns out, however, that for the structures studied so far, nothing changes with respect to equilibria. (Specifically, the strategy profile just analyzed is not an equilibrium under either rule.)

5.4 Vendor specialization; asymmetric technologies

Finally, we look at situations where vendors are specialized in certain components, i.e. the cost structures are asymmetrical. In the following cost structure, each vendor is primary specialist in one product and one component, and secondary specialist in another product and another component. But usually, a vendor specialized in a given product is not specialized in the components this product needs, which should encourage cooperation:

Vendor	a	b	c	v	x	y	z
1	40	10	20	50	5	10	50
2	10	20	40	50	10	50	5
3	20	40	10	5	50	50	10

Table 9: Asymmetrical development costs

	q_1	q_2	q_3	w_1	w_2	w_3	$\sum w$
1.	c-xy	a-z	ab-vz	155	15	85	255
2.	c-x	ab-vxyz	-	45	165	0	210
3.	c-x	a-xyz	b-vz	45	105	75	225
4.	ab-vxyz	ac-xyz	-	45	35	0	80
5.	b-vyz	a-xyz	ac-x	$56\frac{2}{3}$	$11\frac{2}{3}$	$36\frac{2}{3}$	105
6.	b-yz	ab-vxyz	ac-vx	$6\frac{2}{3}$	$11\frac{2}{3}$	$61\frac{2}{3}$	80
7.	b-y	ab-vxyz	ac-vxz	$56\frac{2}{3}$	$11\frac{2}{3}$	$51\frac{2}{3}$	120
8.	b-y	ab-vxz	ac-vxyz	$56\frac{2}{3}$	$21\frac{2}{3}$	$41\frac{2}{3}$	120
9.	-xy	ac-xyz	abc-vz	$43\frac{1}{3}$	$18\frac{1}{3}$	$103\frac{1}{3}$	165
10.	-	ac-xyz	b-vz	0	135	75	210

Table 10: Nash equilibria for asymmetric case

Equilibrium 2, 3, 4 and 10 are autonomous. Equilibrium 1 is a hybrid licensing / threatening-to-license equilibrium similar to the one described above. Vendor 1 licenses y (which he is now specialized in) to either vendor 2 or vendor 3 (exclusively, in the interest of total revenue). Equilibrium 5 is a classical threatening-to-license equilibrium. Equilibrium 6 is a new special case of threatening-to-license, since both vendor 1 and 3 need each others' components - they both produce components that they do not need for their own products, because they can make these components cheaply, and

they both lack components that would be expensive to make themselves. Our common sense expectation would be for these vendors to cooperate, but the anti-trust rule used here is "not strong enough": Since vendor 2 could in principle license v to vendor 1 (in fact, this may be the only reason why he makes this component), he is allowed at the negotiation table, and no components are licensed in order to maximize the total revenue for all three vendors. Equilibria 7, 8 and 9 are similar cases.

This cost structure therefore only leads to limited licensing: One equilibrium where licenses are actually sold as well as leveraged as threats, and several cases where licensing would likely occur under stronger anti-trust rules.

In the final example, cost structures are even more strongly asymmetrical:

Vendor	a	b	c	v	x	y	z
1	30	150	150	150	20	20	150
2	150	20	150	150	150	150	20
3	150	150	20	20	150	150	150

Table 11: Strongly asymmetrical development costs

This leads to the following pure Nash equilibria:

	q_1	q_2	q_3	w_1	w_2	w_3	$\sum w$
1.	a-xy	b-z	c-v	$41\frac{2}{3}$	$106\frac{2}{3}$	$81\frac{2}{3}$	230
2.	a-xy	-z	c-	85	70	45	200
3.	-x	-	c-	45	0	45	90

Table 12: Nash equilibria for strongly asymmetric case

In this case, all equilibria are licensing equilibria. The asymmetries are strong enough that vendors will rather develop the components that they can make cheaply than those that they need for their own products.

References

[Bar92] Doug Barney. Lotus consulting service well received; corporate clients praise effort to integrate technology. *InfoWorld*, Sep 7 1992.

[BN96] A.M. Brandenburger and B.J. Nalebuff. *Co-Opetition*. Harper Collins, 1996.

[Con] The console database. Available from: `http://www.consoledatabase.com`.

[CR01] Neil A. Campbell and Jane B. Reece. *Biology, Sixth Edition*. Benjamin Cummings, 2001.

[CWI00] Aufwertung des microsoft-systems durch it-factory; gröSSter lotus -partner will exchange 2000 unterstützen. *Computerwoche*, page 22, Nov 3 2000.

[CWK01] Krise bei gedys, teamwork und intraware; webifizierung macht lotus-partnern zu schaffen. *Computerwoche*, pages 13–14, Jan 5 2001.

[CWS98] Standardisierung mit hürden. *Computerwoche*, page 27, July 3 1998.

[CZW05] Anwender halten ihren notes-systemen die treue; ibms workplace ist vielen unternehmen zu teuer, zu komplex und zu aufwändig bei der integration. *Computer Zeitung*, (51-52):15, Dec 19 2005.

[GC02] Annabelle Gawer and Michael A. Cusumano. *Platform Leadership: How Intel, Microsoft, and Cisco Drive Industry Innovation*. Harvard Business School Press, 2002.

[GP02] S. Gallagher and S.H. Park. Innovation and competition in standard-based industries: A historical analysis of the u.s. home video game market. *IEEE Transactions on Engineering Management*, 49(1):67, 2002.

[Hei99] Lotusphere: It factory will domino-standards setzen. *Heise News*, Oct 25 1999. Available from: `http://www.heise.de`.

[Kee90] Patricia Keefe. Trials and errors. *Computerworld*, April 16 1990.

[Lot03] The history of notes and domino, Sep 29 2003. Available from: `http://www.lotus.com/ldd/whatisnotes`.

[MS03] David G. Messerschmitt and Clemens Szyperski. *Software Ecosystem : Understanding an Indispensable Technology and Industry*. The MIT Press, 2003.

[Por79] Michael E. Porter. How competitive forces shape strategy. *Harvard Business Review*, March/April 1979.

[Por80] Michael E. Porter. *Competitive Strategy*. The Free Press, 1980.

[Sch95] Jeffrey Schwartz. Lotus blooms after ibm takeover - big blue's deep pockets help the developer pull off a new pricing strategy for notes. *InternetWeek*, 1995.

[Som98] Wolfgang Sommergut. Interessenkonflikte prägen deutsche notes-user-konferenz; lotus verunsichert anwender und partner. *Computerwoche*, June 26 1998.

[Som99] Wolfgang Sommergut. Standard für notes-anwendungen; lotus übegibt lsa an dänischen business-partner. *Computerwoche*, page 12, January 22 1999.

[Wal92] B.H. Walker. Biodiversity and ecological redudancy. *Conservation Biology*, 6:18–23, 1992.

[Web00] Volker Weber. Produktneuheiten auf der lotussphere europe; lotus verschiebt raven auf nächstes jahr. *Computerwoche*, page 22, Oct 13 2000.

[Web01] Volker Weber. It factory implodiert. *Heise News*, 13.12. 2001. Available from: `http://www.heise.de`.

[Weg05] Jürgen Wege. Ceo of group technologies ag and president of german (lotus) notes user group from 1998 until present. interview by the author, June 7 2005.

.