

The Software and Services Challenge



Information Society
Technologies

Contribution to the preparation of the
Technology Pillar on "Software, Grids, Security and
Dependability" of the 7th Framework Programme

30 Jan 2006

Ver. 1.1

Foreword by João da Silva

Software and Services in a Networked World

This report has been written by a group of experts who make clear their view that we have to rethink the technological challenges of developing Software and Services in a networked world. The deployment and delivery of e-Services will be the chief means by which the benefits of the Information Society will be experienced by European citizens.

Those services will be enabled by rigorously developed software. They will comply with user expectations; and they will be robust and resistant to abuse. They will exhibit a high degree of usability. Not only this, but future e-Services will also harness the immense computational power inherent in networked systems while hiding the underlying complexity from users.

The increasing mobility and connectivity offered by myriads of pervasive and ambient computing devices, together with the availability of wireless communication channels, are fuelling software and services with new possibilities that will have an active role in transforming Europe into the world's most dynamic and competitive knowledge-based economy.

Addressing the software and service challenge will facilitate the deployment of a variety of new and innovative services and will ensure that Europe has the capabilities to develop quality and reliable software indispensable for the economic success of any industrial sector of the future networked world.

João da Silva
Director of Network
and Communication
Technologies
European Commission

Background

This document focuses on identifying the scope and structure of research in the area of Software and Service technologies, in preparation for a single Technology Pillar on Software, Grids, Security and Dependability in the EU FP7 programme.

A high-level working group comprising the following participated in the process of preparing this document. The group was chaired by Prof. B. Fitzgerald. Dr. Carl Magnus Olsson acted as editor.

Reinhold Achatz	Siemens	Vice president Software and Engineering Siemens Corporate Technology
Jan Bosch	Nokia	Vice president and Head of Software and Application Technology Lab.
Dieter Rombach	Fraunhofer Institute	Director IESE
Thierry Beauvais	Thales	Director Research and Technology
Alfonso Fuggetta	CEFRIEL	Professor, University Politecnico di Milano, CEO of CEFRIEL
Jean-Pierre Banâtre	INRIA	Deputy Director European affairs
François Bancilhon	Mandriva	CEO
Stefano De Panfilis	Engineering	Director, Research and Development Lab.
Frank Bomarius	Fraunhofer Institute	Director of operations IESE
Heikki Saikkonen	Nokia	Research Fellow (Nokia). Professor of Computer Science, Helsinki Univ.
Herman Kuilder	Thales	Director Software Research, Technology and Innovation
Guenter Boeckle	Siemens	Project Manager
Brian Fitzgerald	University of Limerick	Professor in Global Business and Technology
Carl Magnus Olsson	University of Limerick	Assistant of B. Fitzgerald

Table of contents

FOREWORD BY JOÃO DA SILVA

BACKGROUND

1	INTRODUCTION	2
2	ACHIEVING THIRD GENERATION SERVICES	4
3	RESEARCH AND TECHNOLOGICAL DEVELOPMENT CHALLENGES	6
3.1	SERVICES SCIENCE	6
3.2	SOFTWARE ENGINEERING	7
3.2.1	<i>Software Methods and Processes</i>	7
3.2.2	<i>Requirements Engineering</i>	8
3.2.3	<i>Quality and Reliability</i>	8
3.2.4	<i>Software Product-Lines</i>	9
3.2.5	<i>Security, Dependability and Trust</i>	10
3.2.6	<i>Service-Oriented Architecture</i>	11
3.3	INTEROPERABILITY,	12
3.3.1	<i>Open Source and Open Standards</i>	12
3.3.2	<i>Semantics</i>	12
3.3.3	<i>Technical Infrastructure</i>	13
3.4	COMPLEX SYSTEMS	13
3.5	BUSINESS MODELS	13
4	CONCLUSION	15

This document represents the views of its authors and not necessarily those of the European Commission or its officials.

1 Introduction

The most famous challenge in mathematics for the past 400 years has undoubtedly been Fermat's Last Theorem. Deceptively easy to state,¹ it defied solution for centuries until finally solved by Andrew Wiles in 1994. While the fine details of the proof are not relevant here, the important point to note is that it was not solved by virtue of any radical breakthrough in mathematics; rather, it was through the simultaneous integration of results which had occurred in several disparate and quite unrelated mathematical sub-domains (algebraic geometry, Galois theory and Hecke algebras).

In the ICT field, a similar scenario may be at play. When the *Financial Times* (10 November 05) noted that there were no new buzzwords in ICT, this does not necessarily denote a period of stagnation in which there cannot be any radical innovation. Instead, the lesson from the above is that there can be much to gain through integrating and rationalising what has been happening in the ICT domain in general and in software and services in particular.

Two significant trends towards maturation can be observed in the software and services domain over the past few years. Firstly, in the software field, later binding in relation to actual software deployment has been an important enabler for service adaptability (a later binding time implies better and more flexible and timely adaptation to users' needs). This ranges from pre-deployment binding at compile-time, through binding at installation-time and post-deployment run-time binding, and eventually to ad-hoc fully dynamic binding. This is represented by the horizontal axis in Fig 1 below. Similarly, in relation to services, there has been a progression in terms of service enabling and composition. Initially, this involved simple wrapping of legacy components as services. The next phase involved vertical integration of services by a single vendor, leading on to cross-vertically integrated services. Eventually, the goal is to enable fully dynamic, user driven ad hoc composition of services. This service maturation is represented on the vertical axis in Fig 1.

¹ Fermat's Last Theorem: There are no non-zero integers x , y , and z such that $x^n + y^n = z^n$ in which n is an integer greater than 2.

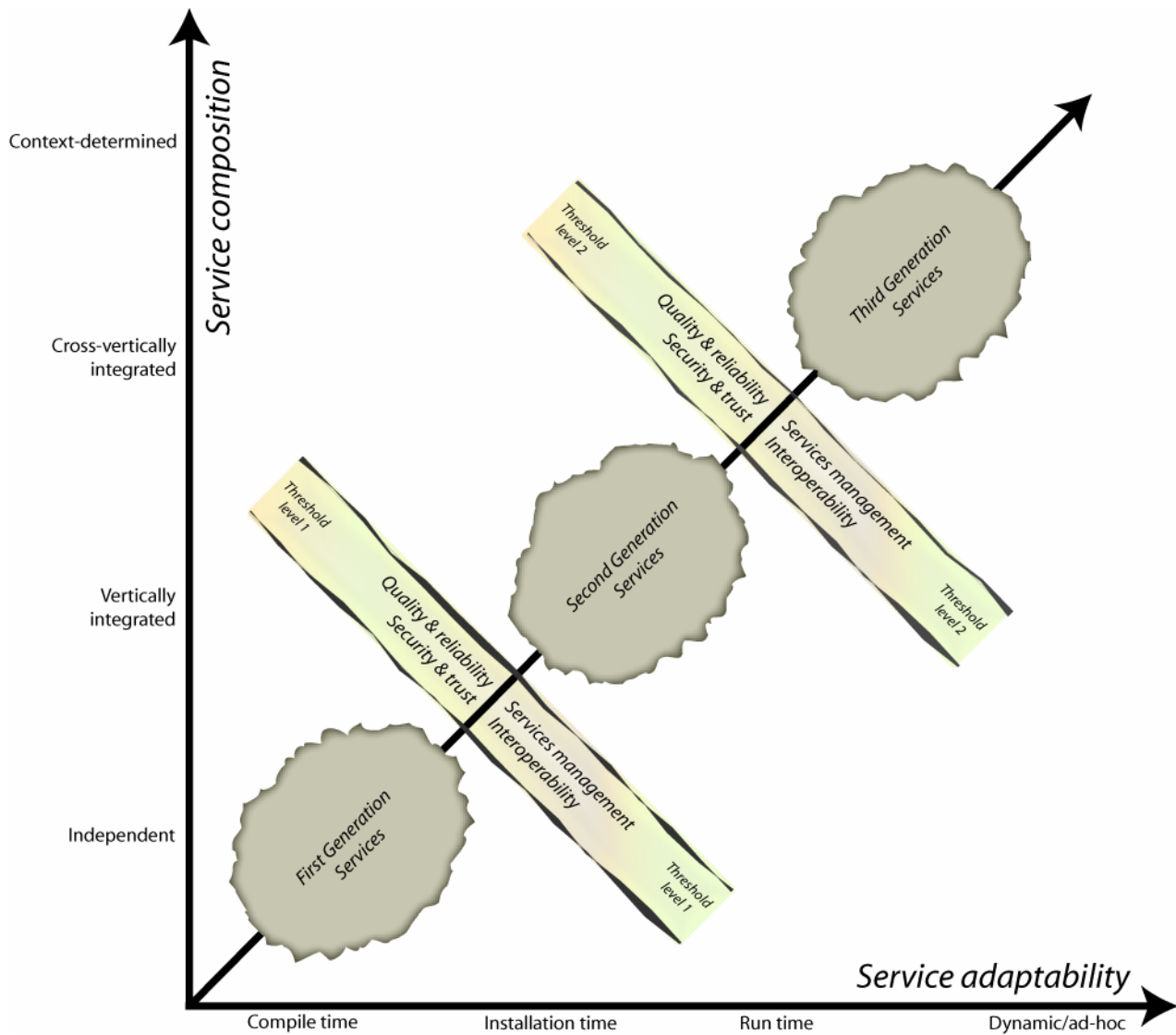


Figure 1: Maturation of software & services

Fig 1 suggests a characterisation of the software and services domain in the past, present and future. The area in the lower left quadrant of Fig 1 reflects the basic state of software and services in the past. This basic level – first generation services – was characterised by independent non-integrated services, much in the manner of once-off conversion of legacy systems to a service delivery mode, or perhaps a service whereby the latest stock market quotes are routed to a customer’s mobile phone. In this basic scenario, the requisite level of services management, quality, reliability, interoperability, security and trust can be quite low.

Moving up from this, we have the case of a plurality of vertically integrated services, where a single vendor bundles a series of related services, for example. This reflects the state-of-the-art in services (second generation services in Fig 1) perhaps, with some successful exemplars of this model. However, in progressing from the previous level to this one, the stakes have shifted in terms of an increase in the threshold level of quality, reliability, interoperability, security and trust, and also in the level of services management to deal with service lifecycle issues, quality of service (QoS) and service level agreements (SLA). A modest variation at this level is to offer cross-vertical services in different domains.

The desired state for software and services is represented by the top right of Fig 1 (third generation services). This level is concerned with context-determined, consumer-driven, dynamically composed services. Mere enabling and integration of services alone is not sufficient. There is a dramatic increase in the threshold level of services management, quality, reliability, interoperability, security and trust required across the heterogeneous enterprises involved. Also, the term 'context-determined' replaces the more commonly used term 'context-aware' to emphasise a more active stance on how software and services should help users create new and meaningful knowledge through the use of third generation services.

The increasing mobility and connectivity offered by myriads of pervasive and ambient computing devices (from portable computers and PDAs to advanced sensors and RFID tags), together with the availability of wireless

communication channels, are fuelling software and services with endless new possibilities for user driven composition of services that have an active role in transforming Europe into the world's most dynamic and competitive knowledge-based economy.

Europe has clearly achieved a leading position in the secondary and embedded software sectors – avionics, automotive, consumer electronics, pharmaceuticals and telecommunications. While this may have been achieved at the expense of establishing a significant presence in the primary software sector, it is interesting to note that the move to software and services blurs the demarcation between the primary and secondary software sectors. Thus Europe can leverage its competitive advantage in the secondary software sector to quickly redress the balance, and achieve overall dominance in the converging software and services sector.

2 Achieving Third Generation Services

While there may be no new ICT buzzword upon which to argue for a paradigm shift, the above is by no means a simple progression. This maturation implies a huge increase in the level of complexity across a range of issues. The challenges of integration, software quality, robustness of service delivery and seamlessness of access across a myriad of platforms and devices, security and trust across heterogeneous enterprises begin to illustrate the complexity of the challenges faced.

Also, while terminology may not have changed in the domain, the actual connotation imbued in the terms used is substantially different. For example, the concept of components was proposed in the seminal NATO Conference on software engineering in 1968, although the concept was expressed in terms of sine and cosine functions, which is far from the connotation associated with components today. Likewise, the term integration connotes a qualitatively

different expectation of what is required in comparison to 10 years ago. Similarly, Parnas's foundational concept of information-hiding in the late 1960s has been quite significant in relation to the fundamental object-oriented concepts of encapsulation and inheritance in subsequent decades. Latterly, the information-hiding concept has broadened to 'transducer hiding' in the automotive sector, and is proving to be a key concept in determining the allocation of development tasks in the current move to globally-distributed software development. In Fig 1, the future state of software and services will require similar complexity-hiding to mask the complexity of integration across multiple domains. This challenge cannot be overestimated. Users of services will demand efficient streamlined easy-to-use composition of services, but providing such ease-of-use represents a hugely complex challenge across a range of areas.

Additionally, the definition of services in the early days (first generation services in Fig 1) has quite a different connotation in terms of the requisite level of services management, interoperability, quality and reliability, security and trust than that expected of the second generation. This is why definitional issues, which bedevil a fast-moving applied field such as ICT, are especially problematic in the rapidly evolving services area. In our context we are primarily concerned with services that are associated with software. Achieving a universally acceptable definition of the term service is probably an insurmountable challenge in itself – whether it be the W3C definition: *a service is an abstract resource that represents a capability of performing tasks that represents a coherent functionality from the point of view of provider entities and requester entities*” or the SeCSE project’s more wide-ranging definition: *A Service is a particular Resource which is offered by a Software System. A Service has a Service Description, which is comprised of a Service Specification (built by the Service Provider) and some Service Additional Information provided, e.g., by Service Consumers using the Service (e.g., ratings and measured QoS) or by the Service Certifier in order to certify some properties of the Service (e.g., trustworthiness) or the results of monitoring activities.* However, what is of more concern are the implications of a service-oriented software domain in terms of the new roles it reveals – service provider and consumer (who can also be a provider), and challenges such as services * (where * represents issues like quality, management, engineering, composition, integration, description, discovery, delivery, etc).

In order to achieve the maturity levels required by third generation services, a number of strands of research will play a key role. These will cover-disciplinary domains from complex systems and services science to core software research areas such as software quality, software product lines, requirements engineering, software methods and processes, and open source software. Figure 2 below elaborates the research roadmap for third generation services.

There are a number of transformations in worldview required by this maturation. While there is an obvious shift (or merging) from products to services, there are also shifts from reductionism to constructionism (borne out in the complex systems perspective); from development to composition in terms of software services, from efficiency to effectiveness (a quality and reliability issue). Furthermore, there is a significant shift from a user/owner perspective to a more utility-oriented model, similar to that of gas and electricity –that is, a move from the traditional buying of software (which is owned and used without restriction) to using and paying for just those specific services. Furthermore, these services, contrary to traditional software, do not need to run on the users’ own computing device. More likely, these services would run somewhere on the network, invisible to the end-user, using whatever computing power and architecture is necessary. This flexible and dynamic service composition and provisioning could be manifest either by immediate user action (pull), by using explicit subscription or implicit agreement in client-server or peer-to-peer service relationships. Examples are proactive and context aware services and context based search.

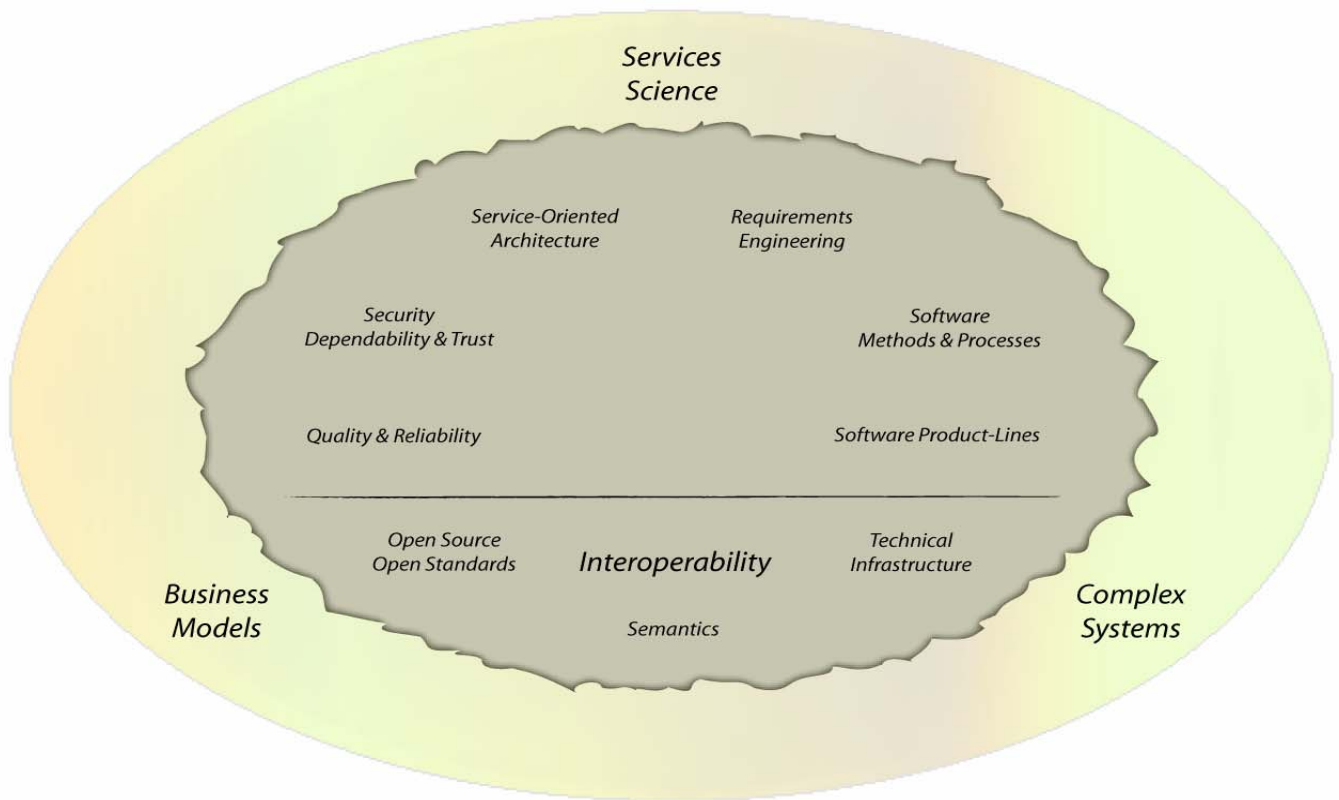


Figure 2: Key elements of third generation services

3 Research and Technological Development Challenges

3.1 Services Science

The inherent complexity of third generation services requires a multi-disciplinary perspective such as that implied by services science. It is frequently assumed that users will simply demand services and drive the enabling and dynamic composition of services. However, we lack understanding of basic issues such as the kind of services users will want and the motivation behind user needs, which is primarily a sociological challenge. There are also business issues in terms of how to charge for such services and reward collaboration by contributors. Services management needs to establish fundamentals such as appropriate service lifecycle mechanisms, SLA and QoS

agreements. Addressing these issues requires a multi-disciplinary perspective drawing on sociology, economics, law, management and marketing etc.

Here the establishment of services science body of knowledge (SSBOK) could be a useful high-level initiative. This would build up a body of understanding about how clients and providers interact differently with service models and product models, the different needs for training, documentation, and evaluation, feedback mechanisms in service-based economies, and different economic and business models for service vs. product industries.

3.2 Software Engineering

3.2.1 Software Methods and Processes

Most software development methods in use today have their origin in foundational concepts that emerged in the research literature in a particularly fruitful decade between 1967 and 1977² – for example, object-orientation, the software development lifecycle, user participation, information-hiding, functional decomposition, prototyping, the structured approach – all can be traced to the literature in this period. However, as already mentioned the current software development context is radically different from that faced in that period. Thus, there is a need to ‘update the clock’ and ground software development methods on concepts and issues more relevant to the globalised context for software and services. Emerging approaches such as aspect-oriented development (AOD) and model-driven development (MDD) need to be elaborated. Also, the surprising success of the open source development model provides lessons that can be incorporated into conventional software development in an ‘inner source’ model. New requirements engineering techniques that describe customers’ needs mapped to suitable services architectures and business models are needed for an appropriate system and service development process. In tandem with this, there is a need for tools for improving the software development process that can enable developers and service providers to measure quality and predict reliability.

One of the areas to consider here is *evolution*; it is related to variability, but has its own aspects. Software evolves and we need to deal with services that may be somewhat outdated but have to cooperate with newer services. We have to provide for the evolution of software, architecture, etc. In a software product line we have one or a small group of developing companies. There, evolution may be dealt with by means of patches or software updates. But how can this be done dynamically and largely automatically when many services are of various ages, use different technology, and run in various evolution states from many different service providers that also must cooperate? Again, as with variability, to allow for the evolution of new services, maintainability of new together with old must be achieved to cope with the added level of complexity facing third generation services.

² Some of these concepts emerged in an earlier period - for example, the systems development lifecycle and object orientation may be traced to the mid-1950s. However, the argument is that they became prominent in the decade, 1967-1977

3.2.2 Requirements Engineering

Requirements engineering (RE) methods, techniques and tools have been conceived in the context of traditional software development, i.e. where the application structure and behaviour is designed to meet the functional and non-functional requirements before the system is deployed and used. Quality control is mostly based on testing systems before they are deployed, and maintaining them afterwards by distributing corrective versions.

In service-based applications, the services that will be used and their quality of service are not selected at design time, and must be separately carried forward as requirements to be placed on services later. Furthermore, part of the requirements need to be expressed or changed at run-time to take into account the adaptable and dynamic nature of service based systems. Checking that services meet the design-time and run-time requirements is not trivial because the mechanisms to use services are complex. Services may need to be searched for, composed and used at run-time. Several services might match the search criteria and might have to be compared to find the one that best meets the needs. Service level agreements might need to be negotiated to agree on the quality of service, conditions of payment, etc. This can be done manually at run-time by the user via high-level interfaces. This can also be done semi-automatically in interaction with the user, letting him or her take the final high-level decisions. This could also be done in a fully automatic manner, which is interesting in the case of machines or devices interacting directly.

From an RE perspective, we need to provide methods, techniques and tools that will allow

3.2.3 Quality and Reliability

The challenge for software (or systems) engineering of modern systems is to handle this change and complexity and still deliver dependable systems with acceptable performance. Current software engineering

us to guarantee that the services used at run-time meet the design-time and run-time functional and systemic (non-functional) requirements. To take into account the dynamic nature of service based systems we must also provide mechanisms and tools to allow users to express new requirements at run-time and enforce them through the service lifecycle, e.g. selection of services, policies or negotiated service level agreements. We also need to realise that future systems and services will, just like today's, have moments when they fail, are attacked, misused, or rely on poor and unpredictable infrastructure. The ability to elicit secure and dependable requirements and understand how to transform them into appropriate architectural solutions, is one of the challenges of third generation services.

Also, we can see a shift from having *efficiency* as primary concern to a focus on *effectiveness*. Operational efficiency through software quality and software development processes (i.e. 'doing the thing *right*') has largely been resolved as organisations can now pretty routinely deliver software on time and within budget. However, the final part of this challenge, that of building a high quality software product, is more elusive. The task of identifying and describing software requirements is known to be difficult, even when all those involved are co-located. In the software services area, stakeholders will be globally distributed across heterogeneous enterprise networks and in this context, a disciplined approach that can ensure quality and reliability is paramount, and thus, requirements engineering is an important research area (i.e. 'doing the *right* thing').

approaches are likely to be too rigid for designing such systems, and the pace of change in the services world is likely to turn into an additional source of system failures. In delivering highly complex and mission or

life-critical services, the threshold level of quality and reliability required is extremely high and hard to achieve. The challenge for software engineering is to develop tools, techniques and methods that will harness the inherent dynamic nature of the world of services to build much more flexible systems, i.e. that can adapt to service changes quickly, and still be dependable. This flexibility cannot be based on tight coupling between services based on stringent specifications. We must instead require flexibility at the service interface level, and build the required adaptability into the systems (autonomic computing). Loose coupling is required at the service interface, for example if a service fails then it needs to be substituted with another similar service whose quality of service can be renegotiated temporarily as long as it still meets the application requirements. Furthermore, it is important to remember that we need methods and tools that are clearly dependable and secure. That is, they explicitly support the construction of systems and services that have properties of security and dependability. We need to be proactive in developing such systems and to guarantee their security, quality of service, reliability and dependability.

To enable robust, end-to-end service contexts with acceptable and predictable service delivery and quality in an environment of varying connectivity and computational resources, we need to build on a robust architecture in the sense that it enables swift adaptations and allows for changes in connectivity and the availability of computational resources. This is equally relevant for large-scale end-to-end resource virtualization from mainframe servers as it is to small devices at the network edge. To

3.2.4 Software Product-Lines

Software product-line (SPL) engineering is a process and set of techniques aimed at making explicit the trade-offs among development interval, quality and cost; ensuring an appropriate balance among them; and embedding the explicit decisions in the architecture and tools used for developing products. Key to SPL is the idea that one is

deliver this, we require features such as service based proactive caching of information, and end-to-end service level adaptation to changes in connectivity bandwidth, latency or available energy.

An important challenge for third generation services will lie in the continued development of Quality of Service (QoS) to fit with service needs. Key to this will be identifying how to elicit, negotiate, document and validate the required qualities of service. This calls for new requirements engineering approaches that consider systemic requirements in the context of service engineering, as well as for the advanced approaches which will ensure meeting these requirements through the development process.

Tools for ensuring QoS include traceability support over various kinds of models and the automatic checking of qualities during development – such as static and dynamic analysis of services and systems that are composed of these services. Illustrating this, tools for the automatic detection of undesired service interactions can contribute to assure the desired level of quality. Besides testing whether the functional requirements are met by a service or a system of services, the desired quality of service has to be evaluated by tools also. These models and tools additionally need to support reasoning about both normal system behaviour and the behaviour of the system dealing with a wide variety of threads which services will be inevitably facing (malicious intentions, infrastructure failures, service incompatibilities, etc). Finally, test cases should be derived in such a way as to allow for the evaluation of the specified quality goals.

always developing a family of products. SPL proceeds by defining what will be common to all family members and what will be variable among family members. The results of such a commonality/variability analysis are used to drive the creation of a modular software architecture that makes it quick, easy, and inexpensive to create products in the family

so long as the commonalities are not violated and the variations in the products stay within the limits of the variability.

Variability thus becomes a key aspect of software product line engineering that will significantly influence other areas. In software product line engineering, the commonalities are captured and realized in a platform. The variability is modelled and defines the space of the current and planned products of the product line. With third generation services, we will have to consider bigger – possibly

3.2.5 Security, Dependability and Trust

At this level, a holistic end-to-end perspective on security that addresses all technological layers is required. Also establishing high levels of trust is critical in the collaboration among organizations operating in virtual environments and the sharing of strategic systems in a global cooperative eco-system. The research agenda here encompasses licences, certification mechanisms, security mechanisms, legal issues, openness as in the open source model, identity management, biometric authentication, fully automated policy-based authorization, trust inheritance, digital rights management and lightweight charging schemes such as micro-payments. The importance of security, dependability and trust cannot be understated, as is reflected in the earlier sections on quality and reliability as well as quality of service development and tools. To give some specific examples, dynamic fault tolerance, composition level error recovery, service fault management, adaptive system resilience, and dynamic damage confinement all need to be handled in third generation service architectures. Common security policies across heterogeneous enterprise networks will also play a role in facilitating common policies for management of security.

even infinite – sets for variability. The provision of theory and methodology for dealing with variability therefore constitutes a major challenge. Modelling and engineering variability will help to cope with the complexity and the huge variety of different services. Support technologies like configuration management that help with administration of the variants need to be extended to cater for generalized variability also, as maintainability of rapidly changing services will undoubtedly be instrumental to the realization of third generation services.

Engineering of such secure and trustworthy services requires advances in development approaches (including tools and rigorous methods) to explicitly deal with fault tolerance, resilience, security, reliability, degradation, dynamic reconfiguration and adaptability of software and services are called for.

As an illustration of how trust and dependability are affected by the third generation service concept, we may consider them from a transaction perspective. If a service S is called by a user, and S will itself call on other services, that in turn will call other services, then we get a cascade of services. All of these may leave traces in the network, e.g. in servers on their route. We now have two possibilities:

1. S completes as requested by the user. In this case we have to make sure that changes occur only at the places specified by S (e.g. data changes), and that all other places are cleaned up, i.e. all user data transported by S and stored in the network and data that may allow conclusions about such user data must be deleted.
2. S does not complete as requested by the user. Then all changes made anywhere in the network must be undone to the state before S has been issued.

From this transaction perspective, third generation services need to deliver this form of guarantee to achieve trust from citizens, companies or public administrations in using software services. Thus, a service called by a user either completes correctly and does not leave any trace behind (except the user-specified one), or it does not complete correctly but no 'half-finished jobs' are left hanging around. In the case of database systems, a specific theory of transactions and its design and implementation has been developed. For third generation services, there will be a similar need that must be satisfied.

3.2.6 Service-Oriented Architecture

One of the key enabling concepts in the software and services area is that of service-oriented architecture (SOA) which facilitates the exposure of independent, loosely coupled and highly interoperable services. Given the enormous assets invested in legacy code and in the education and training of software developers, there is a critical need to identify and re-engineer existing systems to use an SOA approach. SOA emphasises software modularity, extensibility and interoperability, but successful SOAs must be applicable across the spectrum of system granularity and performance. In SOA, modularity is achieved by designing applications as software components. Flexibility can be maximised by the dynamic discovery and on-the-fly network binding of the needed components. Interoperability relies on standardised protocols and on infrastructures capable of adapting differences in the capabilities expressed in those protocols. In addition, standardisation of interaction protocols and interfaces plays a big role in allowing heterogeneous entities to collaborate by making use of substrates that

Moving on, as dependability and trust are also closely linked to the payment methods involved with service provisioning, we need to ensure that our business models are aligned with the technical implementation and protect the end-user from possible security breakdowns. Only through this alignment can end-users perceive the trust needed to rely on a service-based environment. Equally important is the business perspective of this, as the charging mechanisms for service provisioning need to protect the interests of the service provider. This will become an important challenge to be overcome, particularly for peer-to-peer networks

are platform and transport-layer agnostic. But for SOA to be successful, these components must be expressed in a way that allows them to be bound early when performance is critical or late when flexibility is paramount. Costs of developer education and training also dictate that the software cannot be written differently to accommodate different binding requirements.

As an architecture, SOA is domain-agnostic, but relies on different instantiations of the architecture to build concrete systems. Experience at engineering flexible, service-oriented systems indicates the need to be able to take advantage of earlier binding times in order to obtain acceptable performance. The need to accommodate a range of variation in choice of binding-time/performance with a single software component poses the greatest challenges to component software development, and has impact on the programming languages, tools, and middleware infrastructures that connect them.

3.3 Interoperability,

3.3.1 Open Source and Open Standards

Although recent media attention in open source software (OSS) has focused more on desktop applications, the origins and strengths of OSS have been in the platform-enabling tools and infrastructure components that underpin the Internet and Web services – software like GNU/Linux, Apache, Bind etc. In the current shift to network-centric software and services, OSS is likely to become the overarching software engineering paradigm.

Open standards have a number of significant benefits, including increasing privacy and transparency of user data, and reducing dangerous situations in which commercial software vendors exploit proprietary data formats to consolidate and augment their market share. Open source software is linked with open standards. Leveraging the advances within open source is also likely to provide great opportunities for large scale, cross-industry establishment of (an open-source-based) infrastructure using open standards. Especially for services, with the diversity of levels they work on and with the need to combine various services, we need modelling and development support for interoperability so that a peer-to-peer (end-to-end) perspective is possible. This will help to deal with the rapidly increasing complexity

3.3.2 Semantics

Semantics serve to empower the consumer in making sense of the problem context. There is a need for description of processes, services and related information in a form that can be used for service discovery, composition, etc. to implement and deploy complex business processes. The description should take into account context and culture. As systems become of larger scale, encapsulating a larger user base, cultural aspects will play a larger role in third generation. User interfaces and issues of privacy will have to be designed in a flexible way to support these diverse cultural and legislative aspects. Again, to achieve run-time integration, support for dynamic and largely automatic evolution of services will be required. Of course, mediating the emerging

that will be exacerbated by having to deal with several levels at once.

Software and services need to exhibit seamless integration across the boundaries of loosely coupled heterogeneous enterprise networks. A common abstraction language supporting a third generation service architecture would ideally allow itself to be used effectively on all kinds of user terminals. As mentioned earlier, this would also need to support development based on composition and a run-time integration of services. Interoperability can also be achieved through strategic management of the evolution of existing protocols (HTTP, SOAP, TCP/IP) and standards (UDDI, WSDL, XML) to address scalability, reliability and performance as these are applied in mission-critical and novel areas.

Enabling the dynamic and seamless configuration and management of large-scale distributed service systems including ambient services and data grids, will require policy based, but highly autonomous setup and management protocols, and solutions for large scale distributed systems including the myriads of small devices in heterogeneous domains.

protocols and standards for ontologies (e.g. business process management) to ensure uniformity and completeness will always play a large role for semantics. Furthermore, defining appropriate levels of service transparency, featuring seamless integration of services across all technological layers that also empower the end-user with control over service behaviour appropriate to the specific context lies within the semantic sphere. The goal is to enable end-users to create services and service constellations they can share with other end-users, perhaps through peer-to-peer networks. Easy-to-use tools and application frameworks that support high usability are needed to allow for the creation of services that in themselves also become third generation services. The

usability and sharing of sensor data from personal mobile devices is one such enabler.

3.3.3 Technical Infrastructure

This concerns platforms, networks and devices. Resources and devices ranging from small embedded devices to huge computing clusters need to be considered here. Different levels of conformance to this are plausible, e.g. basic level services providing data measurements, to high-level services such as complex simulations on parallel computers. Virtualisation, convergence and mobilisation are three key challenges from a technical infrastructure view. To achieve virtualisation, we must provide seamless service connectivity and facilitate clustering that enable grid computing. Convergence relates back to the issues of interoperability, as unification of voice and data communication

as well as computing will have to be provided through robust and resilient secure services. Finally, as more and more mobile and embedded systems surround us, we need to support this mobilisation with location based context-aware and context-determined services. This ambient service infrastructure will thus need to support the sensor capacities of mobile as well as embedded and stationary devices and transmitters, from RFID tags to processing enabled units. Example gateways include personal mobile devices, home gateways, and on-board car computers. The automotive domain in particular will be one in which many of these services will converge..

3.4 Complex Systems

Service-based applications can be expected to be complex, involving many sub-services that evolve over time. Such applications can be expected to be complex networked applications without any central design: an application uses a service at run-time without much knowledge of its internal design apart from its published service specification. Furthermore the services will have to be continuously provided to end users irrespective of the amount or complexity of the underlying service changes.

The move to third generation services requires a shift from an approach based on

reductionism to one based on constructionism. The design and management of real-world services represents a complex systems problem that does not respect the reductionist boundaries of individual research domains. Complex services have outgrown traditional business models as loose networks of companies collaborate to deliver new services in an entirely new services ecosystem. This has been one of the central lessons of the empowered community ecosystems that have emerged by means of the open source phenomenon.

3.5 Business Models

In the long term many believe that a service-based market will emerge and play an important role in our future digital economies. If this is to happen, such a market should be global and open to allow service providers to offer services in a competitive manner, and allow users to go shopping for the services they need. Users should be able to look for, compare, buy and use the services from the

market that best suit their needs. Access to such a market should be uniform for all kinds of users.

Software product and service providers must therefore increasingly change their products and processes, regardless of whether they develop embedded systems, cooperative

information systems, or distributed and open systems. They must respond rapidly to the new technologies and the business opportunities they generate, as well as to changing customer demands. New business models have to be constructed and adapted quickly to customers' wishes in increasingly dynamic virtual enterprises. The prosperity of industry and consequent job creation will provoke a shift from the development and take-up of technology towards the provision and adaptation of business models. Business modelling is closely related to new requirements engineering techniques and tools that provide the link and adaptation between customer wishes, appropriate business models and services architectures. When developing appropriate business models, it will be vital to ensure that there is a willingness to pay for all levels of services – from low-level to high-level.

Elements of security, trust and reliability may also be incorporated into business models for third generation services. Similar to credit card insurances, the complete and utter security of a service may not always be required, as long as the business model supports and protects the end-user from consequences arising through security breakdowns.

The concept of digital business ecosystem (DBE) is relevant here³. The DBE represents a network of co-existing entities operating in a spirit of co-competition, another lesson from the open source phenomenon. The DBE allows businesses to operate more effectively and efficiently using internet-based technologies, with partners aware of each other's services, and success dependent on finding the right combination of partners, who can provide complementary products and services including those to do with legal and contractual issues.

The open source model has caused the emergence of innovative new business models for software and services. Indeed, since the possibility of using product price as a basis for competition has been largely negated in the case of open source software, new players in the space have to compete by adding value or through the provision of new services.

An FP6 Integrated Project is focusing on DBE research (<http://www.digital-ecosystem.org/>)

4 Conclusion

The importance to Europe of fully leveraging the maturation of software and services cannot be overstated. While Europe has established clear world leadership in the secondary software sector, the same cannot be said for the primary software sector. Given that the move to software and services will cause the primary and secondary software sectors to converge, this affords Europe the opportunity to move to the forefront and achieve long-term sustainable competitive advantage. When NEC President, Tadahiro Sekimoto, stated that “whoever will be able to conquer software, will be able to conquer the world”, this was not an exaggeration. Leveraging the maturation of software and services can provide the means by which this may be achieved.

We see today that most of the leading edge features of any product or service in the market are defined by Software. Europe must not fail to develop this enabling capacity for future economic success.

