Contract Number 034051

InterLink

International Cooperation Activities in Future and Emerging ICTs

Coordination Action
Priority 2 Information Society Technologies

Deliverable number D4.1
Road-Mapping Research in Software-Intensive Systems
and New Computing Paradigms

Due date of deliverable: Month 27
Actual submission date: 1 December 2009

Start date of project: 1 October 2006
Duration: 32 months

LMU-MUENCHEN Version 2

Authors: Martin Wirsing, Matthias Hölzl, and Axel Rauschmayer

<table>
<thead>
<tr>
<th>Dissemination Level</th>
<th>PU</th>
<th>PP</th>
<th>RE</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>PU</td>
<td>PP</td>
<td>RE</td>
<td>CO</td>
</tr>
<tr>
<td>Restricted to other programme participants (including the Commission Services)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restricted to a group specified by the consortium (including the Commission Services)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Confidential, only for members of the consortium (including the Commission Services)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Contents

Executive Summary 5

1 Introduction 9
   1.1 Unlimited Connectivity 9
   1.2 Globalisation 10
   1.3 Security 10
   1.4 Embedded Systems 11
   1.5 Monitoring and Control of Large Systems 11
   1.6 Software-Intensive Systems 12

2 New Issues and Challenges 12
   2.1 Scale 12
   2.2 Heterogeneity 14
   2.3 Openness and Change 15
   2.4 Situation and Localisation 15
   2.5 Resource Constraints 16
   2.6 Data 16

3 Proposals for New Research Themes 16
   3.1 Foundations, Modelling, Analysis 19
      3.1.1 Harnessing Parallelism 20
      3.1.2 Unconventional Models 20
      3.1.3 Dealing with Component Failure 21
      3.1.4 Time and Distribution 21
      3.1.5 Data 22
      3.1.6 Dynamic Complexity 23
   3.2 Languages, Compilers, Platforms 23
      3.2.1 Reflective Languages and Platforms 24
      3.2.2 Contextual Reflection 24
      3.2.3 Run-Time Infrastructure 24
      3.2.4 Support for Autonomous Adaptation 24
   3.3 Methods, Tools, Processes 25
      3.3.1 Methods 25
      3.3.2 Tools 25
      3.3.3 Development Processes 26
   3.4 Testbeds and Systems 26
      3.4.1 Physical Ensembles 26
      3.4.2 Societal Ensembles 27

4 Cross-Thematic Group Synergies 29

5 Suggested Instruments for International Cooperation 29

6 Conclusions 30

A List of Contributors 30
Executive Summary

Introduction

Society has an increasing need for information and communication technology (ICT). Almost every system developed today is software-intensive, i.e., it contains large amounts of software that controls parts of the system or communicates with other systems. Current and future developments in software-intensive systems (SiS) mean that the science and technology of SiS has to face the challenges of scale, distribution and adaptation. The following are some socio-economic factors that influence these challenges and are in turn being influenced by more powerful SiS:

1. Unlimited connectivity: Mobile systems and increasing miniaturisation lead to a myriad of interconnected communication and computation devices.
2. Globalisation: The globalised economy is both enabled by and a cause of advances in ICT. Guarantees of reliability become business-critical for companies.
3. Security: Larger networks and more interconnectedness make security an even greater challenge than it has been in the past.
4. Embedded systems: Miniaturisation and price decreases of micro-controllers has lead to them being ubiquitous. A new tendency to connect different controllers increases the demands on correctness, trustworthiness and resilience of systems.
5. Monitoring and control of large systems: The trend to software-based control is also evident for large facilities, such as industrial plants or national infrastructure. Many of the functionalities provided by these facilities would not be possible without sophisticated software control.

New Issues and Challenges

The increasing scale and heterogeneity of systems, the openness and variability of their environments, and the increasing incorporation of sensors and actuators lead to the following challenges. Current approaches to software engineering either meet these challenges inadequately or not at all.

1. Scale: The tendency to scale systems up to many components is driven by the advantages of scale (performance, fault tolerance, external services), by rapidly shrinking cost, size and energy consumption of traditional hardware components and by the emergence of a number of new technologies and paradigms. Lastly, devices have become more networked which often leads to sub-networks becoming ad-hoc systems. Scaling up reveals the limitations of traditional engineering: microprocessors are hampered by their high power densities, failures of nodes become frequent and must be adapted to, designs cannot be centralised any more and software updates have to be installed dynamically, while the system is running.

2. Heterogeneity: Technological progress has lead to a broad range of devices specialised for different applications or usage scenarios. As these devices are integrated into systems, systems become more heterogeneous. Computation power and network bandwidth of different nodes vary by several orders of magnitude, leading to many different scenarios that need to be implemented.

3. Openness and change: Traditionally, delineating a system boundary has been an important step in developing systems. Many of today’s systems no longer have clearly defined boundaries. They rely on external components for essential services; they contain components that are not fully trusted or that have competing interests; and they operate in environments which are largely unknown during design time. At the same time, rapidly changing business requirements mean that systems have to be frequently changed. But doing so often has to be done while the systems are running. Often systems should self-adapt to react to changing conditions.
4. Situation and localisation: More and more systems contain sensors, actuators, and other peripheral devices. For these devices, location is an important constraint. A recent situation-dependent trend are location-based services, which exploit the position of a mobile terminal, e.g. to provide location-specific information.

5. Resource constraints: With growing networks, resource minimisation is important even for purely computational systems. But it becomes essential where systems interact with the physical environment, e.g. when they run on batteries.

6. Data: As more and more data is being collected world-wide, one of the most important technical problems is how to translate that data into information is useful in a wide context of applications. Similarly, more and more data that has to be preserved is only available in electronic form, giving rise to the need for “eternal preservation” of data over long periods of time, in spite of system failures, enterprise reorganisations and possibly even adversarial parties trying to suppress the availability or existence of the data.

Proposals for New Research Themes

Future software-intensive systems will generally exhibit a number of characteristic features:

- Massive numbers of nodes; nodes with complex behaviour, or complex interactions between nodes.
- Operation in open and non-deterministic environments.
- Variable network topology.
- Need for adaptation to new requirements. We call this future generation of software-intensive systems ensembles. Ensemble engineering is the science and engineering discipline of complex, integrated ensembles of computing elements.

WG1 has identified two kinds of ensembles that we consider particularly promising for driving future research:

- Physical ensembles: are intimately connected to the physical world in space and time. They are equipped with sensors and actuators and have to take into account issues of locality and resource constraints. Examples are real-time embedded systems, claytronics, modular robots or programmable matter.
- Societal ensembles: are closely connected to humans, e.g., smart cities, ambient assistance, or global virtual enterprises.

The following sub-sections propose where research effort into ensembles should be directed.

Foundations, Modelling, Analysis

The main challenge of ensemble engineering may be expressed as harnessing the massive scale, stochastic behaviour and adaptation of ensembles. We divide this research topic into the following parts:

1. Harnessing parallelism: Ensembles with many individual nodes are necessarily massively parallel, and most of the nodes in an ensemble will only be able to directly communicate with a limited number of neighbours. We need to research models, engineering approaches and tools that simplify the conceptualisation, design and development of distributed systems with massive amounts of parallelism. Unconventional models (2) are one promising approach, but traditional models can also be extended. One possibility is to shift from explicit models of all interacting components to statistical abstractions. For example, given a moderate system with 1000 nodes and 100 states per
node, one can view it as a parallel composition of state machines. This results in a state machine with \(100^{100}\) states. Instead, it may be more efficient to consider probability mass functions over the individual states and discrete-time Markov chain models for transitions. Other candidates for extension are process calculi, orchestration and choreography languages, and multi-agent systems.

2. Unconventional models: unconventional models are inspired by sources external to computer science, e.g. biology, chemistry, genetics, social sciences or psychology. Especially models based on the social sciences and psychology have only recently been recognised as being applicable to software-intensive systems.

3. Dealing with component failure: Large numbers of nodes usually imply frequent failures, but also the means to delegate the work of failed nodes. Most current approaches rely on explicit handling of failures. For ensembles, handling unforeseen errors is essential. For example, a system could be structured so that the propagation of erroneous results diminishes with the distance between nodes.

4. Time and distribution: Nodes in an ensemble cannot rely on a global clock. Instead, it may be more appropriate to reason about sequences of events. This kind of reasoning needs new mechanisms so that desired properties can be covered while facing the constraints of systems with resource-limited communication channels or computational nodes. Furthermore, the distributed nature of ensembles makes it difficult to apply ensemble-global functions such as state snapshots. Instead of computing exact results, it might be more useful to first compute an approximation and then refine it, as more distant nodes have had the time to contribute.

5. Data: data can be structured or unstructured. Even if data is structured, data formats tend to differ and translating between them is a largely unsolved problem. If data is unstructured, such as unstructured text, video and audio, new challenges arise such as natural language processing or visual recognition. Humans need to understand data and computers might help by automatically generating visualisations and aggregations. Finally, long-term storage of data faces the hurdles of hardware failures, changing data formats, changing human understanding, and sabotage.

6. Dynamic complexity: the complexity exhibited by ensembles is of a very dynamic nature. This kind of dynamism leads to non-intuitive behaviours, long-term stability is difficult to determine and the system behaviour is difficult to influence. Two research approaches to tackle dynamic complexity are to deduce a global specification from local rules and to find local rules that produce a desired global behaviour. Research is likely to be of an interdisciplinary nature, taking into account results from control theory, general systems theory, system dynamics, cognitive systems theory, etc.

Languages, Compilers, Platforms

Current development tools and languages are not well-suited for ensemble engineering. Possible research topics are:

1. Reflective languages and platforms
2. Contextual reflection
3. Run-time infrastructure
4. Support for autonomous adaptation: has to take heterogeneity of nodes into account. Adaptation will likely span different layers of abstractions with correspondingly different models.
Methods, Tools, Processes

1. Methods: Formal methods for specifying, reasoning about, and analysing ensembles will play an important part in ensemble engineering. The main challenge for formal methods is to scale to the size of realistic ensembles. Finding compositional descriptions for systems that cannot currently be described in such a way is one avenue for solving this problem. Formal methods will also have to adapt to the openness of ensembles by dropping closed world assumptions and the need to re-run complete analyses after minor changes. Lastly, the strict “correctness according to a specification” will probably have to be replaced by the looser notion of “fitness for a purpose”.

2. Tools: Future development tools should have much tighter integration of models and source code.

3. Development processes: currently separate design and development from deployment and maintenance. Furthermore, complete control of the system is assumed. All of this will change with ensembles and external service providers.

Testbeds and Systems

To validate the ideas developed previously, they have to be applied to realistic case studies. Testbeds should concentrate on two main kinds of ensembles (Sect. ), on physical ensembles (massively distributed physical systems and the Internet of Things) and on societal ensembles.
1 Introduction

Information and Communication Technology (ICT) is an important driver of economic progress and economic globalisation. The survey “Information Technology and Economic Performance: A Critical Review of the Empirical Evidence” [DGK03] concludes that “At both the firm and the country level, greater investment in IT is associated with greater productivity growth.” The need for continuously improving productivity in a globalised economy, the possibilities offered by modern ICT for world-wide instantaneous high-bandwidth communication, and the ever-decreasing costs of hardware ranging from embedded devices to enterprise-wide servers and supercomputers drive innovations that are transforming the very fabric of our society. Many products, ranging from children’s toys to cars or manufacturing plants, and even national and international infrastructure, are inconceivable without sophisticated embedded computers; our daily lives are transformed by the ubiquitous availability of portable telephones, broadband Internet and ultra-portable netbooks.

Systems containing large numbers of semi-autonomous embedded controllers and mobile devices are—by necessity—becoming increasingly distributed and decentralised. This trend is further accelerated by another development: performance for single-processor cores will slow down or even flatten; instead even classical processor designs now commonly feature several cores on a single chip, novel technologies will allow the construction of systems with thousands or millions of nodes on a single “chip”, but also with failure rates for individual components being orders of magnitude higher than for current conventional systems.

Today’s systems are woven into many aspects of our lives. Hence, the requirements of the systems, and their environment, are constantly evolving to reflect our changing needs. To support this accelerated rate of change, monolithic applications are being replaced with dynamic federations of autonomous and evolving components where design-time specification of subsystem interactions has to be reduced in favour of dynamic discovery and adaptation. This flexibility can only be controlled by software, traditional, hardware-based control mechanisms cannot cope with these challenges. Therefore almost every system developed today is software-intensive, i.e., systems contain large amounts of software that controls parts of the system or communicates with other systems.

The science and technology of software-intensive systems has to face the challenges of scale, distribution and adaptation, that defy traditional systems and software engineering techniques. In the following sections we elaborate on some socio-economic factors that are, in a mutually reinforcing feedback cycle, influencing these developments and being influenced by the availability of more powerful ICT technologies, and we introduce some of the challenges that these developments pose for the future of software-intensive systems.

1.1 Unlimited Connectivity

Our daily lives are increasingly shaped by the availability of a myriad of interconnected communication and computation devices. Just a few years ago, traditional wire-bound telephones and physical mail were by far the most frequent means for remote personal communication. The first expansion was the introduction of the fax machine which eliminated the high latency of physically transferring documents albeit at the cost of significantly reduced quality and bandwidth. Then the availability of high-bandwidth Internet and ubiquitous mobile communication, together with the decreasing cost and increasing miniaturisation of devices such as computers, PDAs and mobile phones, caused an almost seismic change in the way we communicate, and one might even argue, live our lives. Today it seems almost inconceivable that just a few decades ago the time for round-trip communication between Europe and the United States was usually measured in days or even weeks.

For the first time in history, we possess almost unlimited connectivity: the opening hours of a post office have become irrelevant for most of our communication needs, our phones are no longer tied to the place where they are plugged into the wall, the data we can transmit is no longer restricted to crude copies of black-and-white pages printed on thermo-paper. We can send text, images or video; we can...
single-cast or broadcast; we can choose synchronous communication by instant messaging, or we can communicate asynchronously via email. Similarly, the Internet now offers previously unimaginable amounts of data for instant access. The contents of whole libraries are available on-line, either free of charge or on a subscription basis. Cooperatively generated free and open-source content is rivalling or surpassing commercial offers in quantity and, sometimes, in quality.

This trend is accompanied by the popularisation of more sophisticated systems: video, music and television are being integrated in home entertainment centres that use the Internet to offer value-added services, such as on-demand music or film downloads. Interactions with our banks happen, most frequently, via a Web portal, and we can buy and sell almost everything from the comfort of our homes.

1.2 Globalisation

It is no wonder that the possibilities of an interconnected world have also transformed the operation of businesses: communication between business partners or dependencies is instantaneous, even when large amounts of data have to be transmitted; employees “telecommute” by accessing the resources of their company from their personal computer; employees can freely access the internal resources of their company while they are travelling; business processes are becoming increasingly automated, with humans and computers co-operating to achieve the desired outcome. The globalisation of the economy would not have been possible without the advances in ICT that we have seen in the last decades.

Globalisation, in turn, decisively influences the design of software-intensive systems used in business contexts. It is no longer possible to rely on centralised servers or homogeneous data models, instead decentralised systems with federated access to heterogeneous data sources are becoming the norm; mechanisms that system designers have long taken for granted are no longer available in these environments and have to be replaced with new concepts. For example, transactions have been a fundamental building block for reliable distributed systems. It turns out that in loosely coupled federations where individual parties do not respect the authority of a transaction coordinator, this notion is no longer appropriate and has to be replaced with weaker alternatives such as compensations. How can we build reliable systems under these weaker guarantees provided by the environment? How can we build systems that are resilient to changes in their operating environment? These are fundamental questions that will face systems engineers in the years to come.

A long history of failures, even for critical systems, shows that our reliance on software-intensive systems is not matched by corresponding guarantees for their reliability. If our private Internet connection fails for a few hours we experience, at worst, a minor discomfort; if an important stock market suffers a few hours of outage this may lead to major financial losses. We need to find ways to reliably quantify the failure possibility of systems, and the impact that their failure has.

1.3 Security

Given the ubiquity of software-intensive systems, it is not surprising that security, confidentiality and integrity of the data and the systems on which it resides are major concerns. Can we be sure that the data that we transmit over the Internet between our branch offices is not intercepted and read by our competitors? How do we ensure that the web page that purports to be from our bank is not a criminal scam? How do we know that the formula for the medicine we are taking has not been altered by a cracker who gained access to the network of the pharmaceutical company? In a certain sense, these questions are just special cases of the previously mentioned reliability, resilience and quality-of-service concerns. However, the huge number of attack vectors against distributed, networked systems, the relatively low risk for attackers, the potentially large financial gains from successful attacks, and the psychological and economic impact that successful attacks have, require additional emphasis on security issues.

Security researchers have found many ways to address individual security problems. Yet even a cursory look at the relevant security mailing lists will demonstrate that building secure systems is as yet an unsolved challenge and that trusted systems fail with alarming regularity. Many of the security
breaches that are discovered rely not on conceptual weaknesses of the security systems, but rather on implementation errors [Sch00] or on attackers exploiting weaknesses in protocols or “impedance mismatches” between individual components [And08]. Two of the leading researchers in computer security go so far as to claim: “In the past decade, cryptography has done more to damage the security of digital systems than it has to enhance it. [...] the mathematics of cryptography is almost never the weakest link. The fundamentals of cryptography are important, but far more important is how those fundamentals are implemented and used.” [FS03]. Therefore, further security improvements will not solely rely on novel security mechanisms, but rather on a combination of security research and further improvements in the design and implementation of reliable systems and system components.

1.4 Embedded Systems

The increasing miniaturisation and decreasing price of micro-controllers was one of the most important drivers for technological progress in the last decades. The increased flexibility that embedded controllers provide for the system designer have lead to a shift from mechanically controlled to computerised devices. For example, a car from the 1970s is mechanically not that different from a modern car, but chances are that the old car will contain very little electronics except for the radio, while the modern car may contain dozens of computers with responsibilities ranging from control of the brakes (ABS) or the engine, to satellite-based navigation or regulating the interior temperature.

But system development does usually not restrict itself to the use of isolated embedded controllers, it is often advantageous to connect various controllers. For example, in many modern cars the engine control unit, the transmission control unit and the cruise control can share data via the car’s controller-area network. The possibilities that transmitting some of the internal data to wide-area networks are currently being researched. For example, data from a car’s accelerometers and air bags might be transmitted to a public safety answering point in case of an accident.

The increasing tendency to connect different controllers offers many new possibilities, but it also increases the demands on the correctness, trustworthiness and resilience of systems. What happens if a defective controller saturates the network with data? How do we ensure that the vehicle’s GPS coordinates are only transmitted in the case of a breakdown or accident? How do we prevent terrorists from spoofing accident data to saturate the capacity of the emergency medical services?

1.5 Monitoring and Control of Large Systems

The trend to software-based control is also evident for large facilities, such as industrial plants or national infrastructure. Sophisticated SCADA (Supervisory Control And Data Acquisition) systems and DCSs (Distributed Control Systems) are used to coordinate, monitor and control processes and hardware in such diverse areas as electrical power grids, manufacturing plants, chemical or oil refining plants, or sensor networks. Other important areas, such as air-traffic control networks, industrial supply-chain management, or global financial trade, to name just three examples, also rely heavily on software-intensive systems for their daily operation.

Many of the functionalities provided by these systems would not be possible without software control. However, current approaches to software and systems engineering do not scale very well to systems of that size. For example, a number of vulnerabilities have been found in various implementations of SCADA software. The Israeli security company C4 demonstrated that these vulnerabilities were serious enough that attackers might have gained access to the control centre of plants controlled by this software. It is, however not necessary to look at hypothetical examples in order to see the problems of current software-intensive systems: in March 2008 a nuclear power plant in the USA executed an emergency shutdown after a software update was installed on a computer operating on the plant’s business network; it should never have been possible for a computer in that part of the network to affect the control of the reactor core.
1.6 Software-Intensive Systems

The previous paragraphs show how society is increasingly dependent on software-intensive systems, i.e., systems which interact with their environment and in which software controls the behaviour of individual components and the interaction between components. In future years our dependence on these kinds of systems will only increase, since the unique flexibility that software offers, combined with increasingly powerful and affordable hardware, allows us to build systems that are more advanced, malleable and cost-effective than ever before.

However, the examples also show that we are currently ill-equipped to design, implement and deploy these kinds of systems with the required assurances for reliability, resilience or quality of service. Our current software and systems development approaches are moderately successful for today’s systems, but they do not scale well to the expected size and characteristics of future systems. We are currently lacking the theoretical foundations, the practical approaches and the tools to reliably design and implement future generations of systems.

The InterLink State-of-the-Art Report on Software-Intensive Systems and New Computing Paradigms [HWR07] summarises the state of the art and the current research frontier in many areas relevant for software-intensive systems and forms the technical foundation for this report. The LNCS volume [WBHR08] contains a slightly revised version of [HWR07] and a number of articles by individual members of the InterLink WG1 that present the authors’ visions for future research challenges. This Deliverable synthesises the results of the discussions of the InterLink WG1 and presents them as a unified research roadmap; we have inserted pointers to more detailed or complementary articles in [WBHR08] where appropriate.

2 New Issues and Challenges

Software-intensive systems are characterised by software interacting with other software, systems, devices, sensors and people. Using software to control individual system components and to execute or supervise the overall workflow of a system, offers such significant advantages in flexibility, cost, and performance that today almost every system qualifies as software-intensive.

The omnipresence of software-intensive systems explains their socio-economic importance: many of the amenities of modern life depend on the existence of software-intensive systems, it is not even exaggerated to claim that modern society would not be able to function without these systems fulfilling vital roles. Technological progress will lead to further decreases in the cost, power consumption and size of computers and micro-controllers and thereby further increase the future importance of software-based control.

On the other hand, the increasing scale and heterogeneity of systems, the openness and variability of their environments, and the increasing incorporation of sensors and actuators also lead to a large number of difficulties: Current system often exhibit vulnerabilities and flaws that make them assailable to attack or brittle in the face of unforeseen operating conditions. Systems are too expensive to build and many projects are not completed in time and on budget. Requirements changes often force an expensive redesign and redeployment of the whole system. Because of the importance of software-intensive systems we need to address these challenges. The scope of the problem asks for broad cooperation between European and international researchers.

We will address the main challenges for engineering software-intensive systems in the following subsections: scale, heterogeneity openness and change, situation and localisation, resource constraints, and data.

2.1 Scale

The most obvious trend in the development of software-intensive systems is the rapidly increasing scale of the systems. This development is driven by the numerous advantages that assembling nodes into
systems offers, and enabled by several technological developments.

Some of the advantages of assembling nodes into systems can be demonstrated by comparing a stand-alone computer with a network of computers. By being networked, users can access data on other computers and communicate with users on other computers, tasks can be distributed to several computers to increase the performance, and the failure of an individual computer usually does not affect our ability to continue working in the network. If the network is connected to the Internet these advantages become even more obvious since then additional resources and services become available that are provided by third parties. For example: search machines, online encyclopedias or email, to name just a few.

One of the technological enablers of large-scale systems has been the rapidly shrinking cost, size and energy consumption for traditional hardware components and the emergence of a number of new technologies and paradigms. The flexibility of computerised control means that systems can be scaled up more than before, the cheap price of components ensures that the systems remain cost effective, the reduced size and power consumption of hardware enables designers to build miniaturised and portable devices. A wide variety of components allows system designers to tailor the implementation of a system precisely to its requirements, thereby optimising factors such as cost and energy consumption. A more detailed overview of these important technological developments and constraints is presented in [HWR07, We06].

At the same time, these developments have also revealed that the traditional ways of scaling up systems has inherent limits. For example, the miniaturisation of semiconductor manufacturing processes allows us to build microprocessors containing enormous numbers of transistors; however, it is becoming more and more difficult to use the additional transistors to increase the performance of a single processor, e.g., because of the high power densities [VF05] that such a design implies. This has already led to the wide distribution of dual and quad-core processors; dozens of cores are now already seen on special-purpose processors and prototypes. Even hundreds or thousands of cores on a single chip are just a few years away or, if we count the shaders on graphics processing units as simple CPUs, are already on the market. Further progress in the development of nanotechnology [MS03, Deh05] may lead to systems with millions of nodes on a single “chip.”

Another important factor in the increasing scale of systems has been the deployment of ubiquitous broadband network access. This put geographically dispersed systems in reach of organisations and even individuals who could previously not afford the cost of a dedicated wide-area network.

Therefore thousands of computational nodes with broadly differing capabilities and performance characteristics, operating asynchronously, and connected by various network and interconnect layers are no longer the sole prerogative of the largest, most elaborate systems being built. Even moderate systems, built from commodity hardware can, and often do, exhibit these characteristics.

The increasing number of independent nodes and the broad geographic distribution pose a number of problems for the system designer: in a system containing millions of nodes, the failure of individual nodes cannot be seen as an extraordinary event, instead it will be a common occurrence even if individual nodes have a relatively high mean time between failures. The system has to recognise these failures and work around them, e.g., by re-doing the computation that was taking place on the failed component and by removing the defective node from the system.

Traditional systems often rely on centralised parts to ensure consistency, e.g., to coordinate transactions or to back up the system data. With increasing distribution, reliance on a central component becomes prohibitively expensive or even impossible: in general it is not even possible to acquire a globally consistent snapshot of a system’s state, let alone centrally coordinate the operation of the system. Instead the designer has to find ways to harness this parallelism while ensuring that the system does not suffer from race conditions or deadlocks.

If a system consists of a large number of geographically distributed nodes it is impossible to stop the system to update or replace individual components. Instead, updates have to be propagated through the system while it is operating. Therefore the system has to be able to operate in a state which is only locally consistent while preserving overall correctness. This problem becomes even more pronounced with...
increasingly open system boundaries, where components outside the system are not under the control of the system’s designer and may cease to operate or change their behaviour without notifying the system.

2.2 Heterogeneity

Technological progress has not only lead to more powerful devices, but also to a broad range of different devices specialised for different applications or usage scenarios, for example wireless sensors or portable phones. As these devices are being integrated into systems, the systems grow increasingly heterogeneous, with the computational power and network bandwidth of different nodes varying by several orders of magnitude. This poses additional problems for systems developers that can be illustrated by the following example (taken from [HWR08]). We consider a very simple, if artificial, problem: communicate all prime numbers in the interval $[m, n]$ from actor $A$ to actor $B$. There are various well-known algorithms for solving this problem [Kra87], e.g., node $A$ might use the Sieve of Eratosthenes or a probabilistic primality test to compute all primes in the range $[m, n]$ and then transmit this list to node $B$.

In the description of the solution we have made several implicit assumptions about the system in which we are operating that may not be correct: we have assumed that the network capacity between the nodes is sufficient to transmit the list of primes and that node $A$ is sufficiently powerful to compute the solution. In Figure 1 we have depicted different scenarios that may occur instead: Fig. 1(a) represents the original situation: two workstations connected by a high-bandwidth network. In Fig. 1(b) two workstations are connected by a low-bandwidth connection. In this case it may not be feasible to transmit a long list of primes to node $B$, instead it might be necessary to transmit a program to compute the primes. In Fig. 1(c) node $B$ represents a powerful cluster where it would be advantageous to use a parallel algorithm. It might also happen, as depicted in Fig. 1(d) that the program is running on a mobile phone that does not itself possess the resources to compute the desired solution. In that case an appropriate solution would again be to transmit the program to compute the primes to node $B$. However, if the situation is
as in Fig. 1(e) and two devices with limited resources are communicating, then this solution is no longer applicable. Instead the devices might try to discover another node $C$ that can perform the computation on their behalf and transmit the results to $B$.

2.3 Openness and Change

Traditionally the commitment to a system boundary, i.e., clarifying which components are part of the system and therefore have to be designed, built and maintained by the system developers, has been an important step in the development of systems. Many of today’s systems no longer have clearly defined boundaries; instead systems rely on external components for essential parts of their functionality, contain components which are not fully trusted or which have competing interests, and operate in environments which are in large parts unknown during design time.

For example, some commercial search engines currently index more than a trillion pages; it would be wasteful and prohibitively expensive for system users to operate their own Web search engine with the same coverage. Instead, most systems rely on offers from companies such as Google or Yahoo to perform Internet searches. The system designer often has only very limited influence on the quality of service of components maintained by external providers, and may often not even be notified in changes of their algorithms or service contracts. E-commerce systems for B2B (business-to-business transactions) may link enterprises which cooperate in certain areas while competing in others; and they have to survive the numerous reorganisations and acquisitions that are the norm in today’s business environment.

At the same time, the requirements of businesses in the global economy are changing at a rapid pace; constantly adapting to the needs of the market and customers is an essential requirement for global enterprises. Traditional systems are not well suited to this kind of change: they rely on statically connected components with well-defined functionality, and on a predetermined and mostly deterministic control flow throughout the system. In many existing systems reconfiguration is a disruptive operation that often requires careful planning and planned downtime of system parts.

An important goal for future systems is to better support change: it has to be easy to adapt systems to changing requirements or environmental conditions while the system is running. In many cases the system should self-adapt if it detects changes in its environment that either derail its current mode of operation, or that offer possibilities for improved operation.

2.4 Situation and Localisation

For systems operating on a global scale network topology, bandwidth and latency are often more important than the physical location of the individual nodes. However, with the increasing number of networked embedded controllers and the connection of purely virtual systems to physical devices we are seeing more and more systems containing sensors, actuators and other peripheral devices. For these devices, location is an important concept: sensors in wireless sensor networks can normally only communicate with their immediate neighbours and the interpretation of sensor data depends strongly on the location of individual sensors.

Another aspect that increases the importance of physical situation is the emergence of location-based services as an important business model. The Open Geospatial Consortium defines the term as

**Location-Based Service (LBS):** A wireless-IP service that uses geographic information to serve a mobile user. Any application service that exploits the position of a mobile terminal.  

[Ope]

Examples for location-based services range from simple functionalities such as finding the closest ATM machine or restaurant to complex scenarios such as road billing, fleet scheduling, automotive emergency assistance, or infrastructure management. For these applications, the ability to identify the changing positions of individual nodes and to reconfigure the system according to these locations is paramount.
2.5 Resource Constraints

In spite of the tremendous increases in computing power at our disposal, many systems have to deal with resource constraints and trade-offs. Even for purely computational systems, resource minimisation is important, since the energy consumption of large networks of computers can be staggering. This effect is compounded by the need for air conditioning to remove excess heat produced by large numbers of computers clustered together. But resource constraints are most pronounced in areas where systems interact with the physical environment: sensor networks often have to operate on batteries, and in many scenarios these batteries are difficult to replace after the sensors have been deployed. Therefore, reduced communication overhead or reduction of transmission conflicts can have significant influences on the durability of these sensor networks. For applications such as claytronics [ABC+05, KAG+07], where millions of millimetre-sized nodes physically and computationally interact, efficient utilisation of the available hardware is mandatory.

2.6 Data

As more private and business transactions are conducted online, the amounts of data stored, transferred and processed by software-intensive systems is increasing exponentially. One of the most important technical problems is how to translate that data into information that is useful in a wide context of applications. Similarly, more and more data that has to be preserved is only available in electronic form, giving rise to the need for “eternal preservation” of data over long periods of time, in spite of system failures, enterprise reorganisations and possibly even adversary parties trying to suppress the availability or existence of the data.

One of the benefits of Internet technologies is easy access to, and proliferation of, information. Obviously, the latter can also be a curse; companies and individuals are overwhelmed by all kinds of data: E-Mails, business transactions, blogs, forums, instant messaging conversations, etc. The problem of the data flood has lately grown more pressing for businesses, because—due to regulatory and litigation law—they can be held accountable for the quality of their data.

Data can be loosely split into three groups:

- Structured data: business transactions, contact data, bookkeeping, . . .
- Unstructured text: emails, blogs, forums, . . .
- Binary data, which often comes in large chunks. Examples include images, voice, and video.

While structured data (especially meta-data) is slowly becoming more standardised, its scale pales when compared to unstructured data. To make unstructured data amenable to automation, one has to structure it. Two alternative approaches exist for doing that: On one hand, there is natural language processing (NLP). On the other hand, there is tokenisation, complemented by semantic annotation, clustering or statistical analysis.

Even when data is structured, managing it is still a challenge: Different data formats have to be converted; semantics and schemas have to be translated; communities around the data have to be considered; and data has to be stored (possibly long-term), maintained, and accessed. Privacy also plays a big role; especially in professions that synthesise data from various sources. Examples of such professions are patent attorneys, medical doctors, or tax auditors.

3 Proposals for New Research Themes

As described in the previous sections, future software-intensive systems will generally exhibit a number of characteristic features:

- Massive numbers of nodes, nodes with complex behaviour, or complex interactions between nodes.
We call this future generation of software-intensive systems ensembles. **Ensemble engineering** is the science and engineering discipline of complex, integrated ensembles of computing elements. The potentially huge impact—both positive and negative—of ensembles means that we need to understand ways to reliably and predictably model, design, and program them.

Because of the social and economic importance of ensembles, the difficulty of building them, and the lack of engineering methods that adequately address the challenges posed by future software-intensive systems, we consider ensemble engineering to be one of the most important areas for European ICT research in the coming years.

Many of the problems of ensemble engineering are directly connected to the massive scale of ensembles, the complexity of its components or their interaction, the operation in open environments, and the need for adaptation: How do we design software for ensembles that is reliable, predictable, with guarantees for security and trust, that acts autonomously, has self-* properties (self-healing, self-managing, etc.), and that can harness emergent behaviour? An overview of the proposed research effort is given in Fig. 2, with research spanning the range from foundational topics to the construction of testbeds and systems to verify the proposed methods:

- **Foundations, modelling, analysis.** Massive scale, complexity of components and interactions, stochastic behaviour and adaptation are properties that arise directly from the definition of ensembles. Yet our current understanding of system and software engineering is not sufficient to reliably build software-intensive systems of the required size and with the desired properties. We need formal foundations as well as modelling and analysis approaches that allow designers to build reliable, trustworthy ensembles. Models might be hybrids of stochastic, continuous and discrete elements or take inspiration from paradigms not traditionally associated with the engineering of software-intensive systems, e.g., biology, chemistry, genetics, social sciences or psychology.
ling these foundational problems will be one of the most important challenges in the engineering of software-intensive systems in the foreseeable future.

- **Languages, compilers, and platforms** for ensembles. It is not sufficient to build the foundations for modelling and analysing ensembles, it is also necessary to provide engineers with the languages, compilers and platforms to build them. Languages and platforms need to support reflection and dynamic evolution of ensembles; models of the ensemble have to be available at run-time. Research into languages, compilers and platforms for ensembles is an important foundation for building adequate tools for ensemble engineering.

- **Methods, tools and processes** for ensembles. Our current methods and tools are barely adequate for dealing with current systems. Even the best available tools and languages are only slight improvements over the state-of-the-art from 30 years ago. Development processes and methodologies are mostly based on conjectures and opinions, without scientific studies backing up the claims made by their proponents. Current development tools provide only limited support for distributed development and make no use of the advances in cognitive systems and ambient computing environments. Processes for engineering security-critical ensembles will probably need to include formal methods for reasoning about ensembles that are scalable to the size of real-world ensembles, as well as methods for controlling and adapting ensembles whose subsystems and components are dynamically changing. This may necessitate finding replacements for the notion of “correctness according to a specification” which are more appropriate for the scale and complexity of ensembles. Research about development methods, tools and processes that provide drastically improved support for the developer is sorely needed.

WG1 has identified two kinds of ensembles that we consider particularly promising for driving future research: physical ensembles and societal ensembles. These two classes of ensembles define two points in the landscape of all possible ensemble types: both kinds of ensembles promise to be of great scientific and social relevance, they define ensembles with clearly defined characteristics, and they address two fundamentally different kinds of ensembles.

- **Physical ensembles**, which are intimately connected to the physical world in space and time. They are equipped with sensors and actuators and have to take into account issues of locality and resource constraints. Examples are real-time embedded systems, claytronics, modular robots or programmable matter. These systems combine discrete and continuous, non-linear domains, and they exhibit complex interaction patterns between components. Coordination in space and time with limited resources is one of the major challenges faced by physical ensembles.

- **Societal ensembles** are ensembles that are closely connected to humans, e.g., smart cities, ambient assistance, or global virtual enterprises. Research in this area will have to investigate the dynamics of purposive interactions and how the structure of evolving societal interactions can be reflected in the architecture and the design of the software: evolution of societal ensembles has to be a long-term process that goes beyond single-run adaptation, and systems have to maintain societal coherence while supporting diversity and context awareness.

Testbeds and research prototypes for both physical and societal ensembles should be built as part of future research efforts.

Research on ensembles can be performed in different directions: one method is to use a bottom-up approach that starts with logical foundations and works toward the top of the pyramid in Fig. 2, another method is to use a case study-driven research plan that starts with the construction of a prototype and develops the necessary tools and formalisms based on the experience gained from the development process. The most promising research agenda may be a combination of these approaches, where both bottom-up and top-down approaches are combined, and the development of foundations, languages, tools and processes is informed by, but not completely determined by, case studies. This is illustrated in Fig. 3.
3.1 Foundations, Modelling, Analysis

The main challenge of ensemble engineering may be expressed as harnessing the massive scale, stochastic behaviour and adaptation of ensembles: the large number of nodes in an ensemble and the open and non-deterministic environment in which it operates make it difficult to design the behaviour of the ensemble using conventional engineering methods and increase the importance of dealing with behaviours that can only be described stochastically. The importance of new approaches to systems development is made even more pronounced by the need for adaptation to changing network topology, infrastructure and requirements. This section explores some of the consequences, [HWR08] contains additional information.

Current techniques of system and software engineering are woefully inadequate for designing future ensembles; since the different challenges interact in non-trivial ways we consider it important to focus research on approaches which address a broad range issues rather than on specialised projects dealing with only a single topic.

We divide this research topic into the subsections

- Harnessing Parallelism
- Unconventional Models
- Dealing with Component Failure
- Time and Distribution
- Dynamic Complexity

While each approach to ensemble engineering has to address these properties in some way, they are not exhaustive. Other issues, such as parallel design and development of different system components, or hard- and software co-development, also play an important role. However, we regard the following topics as crucial for the success of any approach to ensemble engineering and therefore important focus points for research.
3.1.1 Harnessing Parallelism

One of the standard textbooks on software engineering [Som07] advises the following: “Parallelism is dangerous because of the difficulties of predicting the subtle effects of timing interactions between parallel processes. […] Parallelism may be unavoidable, but its use should be carefully controlled to minimise inter process dependencies.” This reasoning is reflected in current development approaches as well: they generally rely on the developers to control parallelism and synchronisation strategies, and often the only feasible strategy is to make liberal use of synchronisation and critical regions. Even for systems of moderate size this is an expensive and error prone process, in particular when network failures and timeouts have to be taken into account. For ensembles, their sheer scale makes it impossible to build reliable and dependable systems this way.

Ensembles with thousands, or even millions, of individual nodes are necessarily massively parallel, and most of the nodes in an ensemble will only be able to directly communicate with a limited number of neighbours. Given these premises, restricting parallelism is only possible and useful in rare circumstances, e.g., when proximate nodes try to access the same device; in most circumstances it is either prohibitively expensive or even impossible. Therefore we need to research models, engineering approaches and tools that simplify the conceptualisation, design and development of distributed systems with massive amounts of parallelism. Unconventional models, as discussed in Sec. 3.1.2, are one promising approach to this problem.

However, in many cases it may be more appropriate to rely on extensions of traditional models that allow for greater amounts of distribution and parallelism. There are various forms that these extensions to traditional models might take. One possibility is a stronger shift from explicit models of all interacting components to statistical abstractions:¹ We cannot usefully understand an ensemble as the parallel composition of state machines for individual nodes, since even for a moderate system with 1000 nodes with 100 states each, this would result in a state machines with $1000^{100}$ states. Instead it may be useful to, for example, consider probability mass functions over the individual states and discrete-time Markov chain models for transitions. In this case it may be possible to express properties as inequalities over the probability mass functions and use statistical methods to estimate the aggregate behaviour of the system. Certain properties of systems are most naturally described by stochastic notions, e.g., communication delays in the network structure or component failures. These properties increase in importance with the size of the system and therefore provide another justification for stochastic or statistical abstractions. In particular for ensembles with sensors or actuators, continuous models for their interactions with the physical environment will play an important role, therefore we expect to see research in the area of hybrid stochastic, continuous and discrete models, and also into the quantitative modelling of concurrent systems.

But other possibilities for extending traditional models exist as well, many with their own unique advantages. Models based on process calculi, e.g. PEPA [Hil96, CG08], or orchestration and choreography languages, such as Orc [CM08], may harness the theoretical results and the static analysis techniques and tools that have been developed for these approaches. The incorporation of ideas form multi-agent systems into system design, e.g., by using auctions or negotiations to coordinate individual nodes, may lead to models that allow sophisticated autonomous behaviour and efficient automated distribution of work in the system while still giving the designer enough control to tailor system parameters to requirements.

3.1.2 Unconventional Models

Previously we have addressed the need for new kinds of models for ensembles. While it is possible to model ensembles with extensions of more traditional system models, some unconventional models may offer even greater advantages for resilience, adaptation, controlled emergence or exploiting opportunities.

¹Many engineering approaches in current use take statistical properties into account, however most approaches nevertheless rely too strongly on complete models of component interactions to be directly applicable to ensembles.
These unconventional models for ensembles take inspiration from paradigms which are not traditionally associated with the engineering of software-intensive systems, e.g., biology, chemistry, genetics, social sciences or psychology. Some of these models, e.g., chemical reaction models have already been investigated for more than 15 years and significant progress has already been made in the application of these models to software-intensive systems, more details about research challenges for chemical computing can be found in part III of [WBHR08], in particular [BFR08].

Other kinds of unconventional models, e.g., models inspired by social sciences or psychology have so far not commonly been used in the development of software-intensive systems. However, recent developments in some of these areas have demonstrated results that seem to be applicable for software-intensive systems. For example, in cognitive system theory [Gro08], diffusive control signals, modelled after neuromodulators in the brain, control the “metalearning” processes that are responsible for adaptation. Similar models might result in robust models for adaptive software intensive systems.

The important research challenge in the area of unconventional models is to provide a reservoir of techniques, tools, concepts and methods based on biological, chemical, social, or other principles, to face the challenge of developing ensembles.

3.1.3 Dealing with Component Failure

Having a huge number of nodes in a system forces designers to approach many system properties from a different perspective. One important example is component failure: the probability of component failure in a large system is high, simply because of the sheer number of nodes. On the other hand, the huge number of nodes often allows simple ways to deal with errors, e.g., by redistributing the tasks of failed nodes to operative ones.

Many current approaches provide little support for fault tolerance: foreseeing, diagnosing, and appropriately reacting to all possible faults and errors of components has to be explicitly done by the system designers before the system is deployed, unforeseen errors can have a large impact on the reliability of the whole system. Service-oriented languages often provide compensation activities that are triggered by software errors, but the specification of possible compensations is again a manual process and the possibility of hardware defects is, in general, not entertained. Certain approaches for distributing parallel computation provide built-in error detection and compensation mechanism, but are only applicable for a restricted class of scenarios. One example is Google’s map-reduce implementation which includes built-in support for retrying computations assigned to failed nodes, but which offers only a limited model for concurrent computation.

For ensembles, a more powerful way to handle unforeseen hardware failures or software errors is desirable, if not mandatory. For example, a system might be structured in such a way that the effects of erroneous results which are propagated through the system diminish with the distance between nodes. This kind of system topology would provide strong resilience for the system even in the face of failures and errors that were not foreseen during design time; such general approaches could then be augmented with more specialised mechanism for expected failures. The main challenge in this area is to develop design techniques and tools that facilitate the design and development of systems which are resilient in the face of errors, by mitigating the effects of failures, by automating detection and recovery from failures, or by providing sufficient support for the system developer to provide comprehensive failure recovery even for large systems.

3.1.4 Time and Distribution

One notion that plays a particularly important role in distributed systems is time: nodes in an ensemble cannot rely on a globally unique clock. Synchronising all nodes to a single physical clock source might not be feasible or too expensive, and furthermore, the notion of “wall clock time” is not necessarily the most important aspect in the description of a system. It may be more appropriate to reason about sequences of events [Tal08], and due to network topology or delays the sequences of received events...
may not be comparable at different nodes. For example, if two nodes $S_1$ and $S_2$ broadcast messages $m_1$ and $m_2$ to recipients $R_1$ and $R_2$, it may happen that $R_1$ receives these messages in the order $m_1; m_2$ whereas $R_2$ sees the sequence of incoming messages $m_2; m_1$. Mechanisms for generating partial causal orders of events exist, e.g., vector clocks, but these are generally too expensive to use in ensembles. For example, a vector clock for $n$ nodes consists of an array of $n$ logical clocks; for systems with millions of nodes this becomes prohibitively expensive. Instead, system designers need notions of time that are expressive enough to reason about the desired properties, but inexpensive enough to be implemented even on systems with resource-limited communication channels or computational nodes. One example is the model of a probabilistic cone in the system’s space-time that delimits the area of causal influence that a node can have.

The problem of time and distribution manifests itself in another guise as well: the distributed nature of ensembles often makes it difficult to compute global functions over all nodes of the ensemble. One simple example of this phenomenon is taking a snapshot of the ensemble state: communicating the command to take a snapshot to all nodes in the system takes time; since the system is evolving during this time, the snapshot of different nodes is “skewed” between the start time and the completion time of the snapshot. There is, however, an interesting notion of approximation: nodes that are far apart in the system topology exhibit larger skews than nodes that are close together.

The research challenge is therefore to develop methods that consider and exploit the topology of the ensemble, its distribution in space-time and the locality of nodes to achieve desired behaviour, or to limit unwarranted effects. For example, it may often not be necessary or possible to compute an exact solution to a problem, e.g., if the exact solution takes a long time to compute but the result is only useful before a deadline has expired. A more useful approach might be to try to first compute a rough approximation of the result and then refine this initial answer with input from additional, increasingly distant nodes until the deadline is reached.

### 3.1.5 Data

Systems are processing exponentially increasing amounts of data from a large number of sources. Some of this data is available in structured form; however different data sources often have structured their data in different ways. Synthesising structured data by automatically translating between different schemas or ontologies is still a largely unsolved problem.

But increasingly, data is not available in a structured form, but rather as unstructured text. For example, only a small number of the available web pages provide any kind of semantic markup, most are texts in natural language. Improved understanding of unstructured text is therefore an important research topic: the current state of the art includes word-based analysis via annotators that rely on vocabulary and concept definitions for specialised domains such as biology, chemistry, life sciences, human resources, and competitive intelligence. The challenge is to extend these successes to mixed domains and include abilities such as common-sense reasoning. Understanding video and audio poses an even greater challenge than understanding unstructured text.

Data is much easier to understand for humans when it is aggregated and visualised in a sensible way. Therefore, visualisations are important, but useful visualisations are normally very problem-specific, and inappropriate visualisations can obscure the meaning of the data rather than clarify it. Can the process of finding such visualisations be automated or at least be better computer-supported?

As more and more data is only stored in digital form, the “eternal” storage of data is becoming a pressing problem. However, long-term data storage faces numerous hurdles:

- Hardware deteriorates and fails, physical communication standards go out of style.
- Data formats used in software change over the years; proprietary formats are often undocumented and newer versions of applications often cannot process documents in an older format.
- Cultural references in the data depend on region and time; sabotage and censorship can affect both hardware and software.
Addressing these problems is an important research challenge to preserve commercial investments in data, but also our cultural heritage.

### 3.1.6 Dynamic Complexity

A huge number of nodes is not something that necessarily something that increases the complexity of a system. For example, if the nodes can be arranged in a hierarchical control tree, then an exponential number of nodes can be accommodated with just a linear increase in the depth of the tree, and with no increase in the conceptual complexity of the system.

However, ensembles rarely exhibit such a simple structure, the more common case is that—by necessity—ensembles contain multiple reinforcing and inhibiting feedback loops, the effects of actions are delayed either by network latencies or, more significantly, by elements that buffer or aggregate certain inputs. These features give rise to the high dynamic complexity of ensembles: it is well known from the literature on dynamical systems and system dynamics that these kinds of systems exhibit non-intuitive behaviours, that it is often difficult to determine the long-term stability of such systems, and that it is difficult to find the leverage points from which the behaviour of the system can be influenced. For example, even a system with a first-order feedback loop and a delay element may exhibit oscillations. Dynamically complex systems often show policy resistance, i.e., if they are in a stable state, they show the effect of changes to the system only after a significant delay; this property often leads to over-compensation and oscillations around the desired state. Furthermore, dynamically complex systems often exhibit bifurcations; in the environment of a bifurcation point small changes in input values can lead to radically different behaviours of the system.

For ensembles these effects are compounded by the open environment in which they operate and the variable network topology of the ensemble, as these features make it impossible to determine the ensemble’s dynamic behaviour in advance.

Research that addresses dynamic complexity can take several forms: one possibility is the development of methods to deduce a global specification from local rules; these rules would allow the system designer to show that certain desirable properties do occur or that undesirable properties cannot occur in a given system. However, given the complexity of the problem, this approach can only address a limited range of systems.

The converse method, which may be particularly useful in the design of new systems, is to find local rules that produce a desired global behaviour. This approach allows the system designer to concentrate the objectives of the system and leave the details to the “system compiler.”

Of particular importance to adaptation is research that investigates methods to influence the global behaviour of the ensemble by making local changes. Being able to determine leverage points of the system, i.e., points where a change of the input produces a desired behavioural change in the system, is one of the important prerequisites for further research on adaptation.

However, it seems likely that these research approaches have to be augmented with research about dynamic monitoring of systems, recognising the occurrence of undesirable results or operating conditions, and dynamically identifying methods to adapt the system structure to counteract erroneous system behaviour and achieve desired system outcome. This research is likely to be of an interdisciplinary nature as it will probably take into account results from control theory, general systems theory, system dynamics, cognitive systems theory, etc.

### 3.2 Languages, Compilers, Platforms

Our current development tools and languages are not well-suited for ensemble engineering. Therefore, the research efforts described in the previous subsection should be complemented by research on development tools and languages. Some of the challenges in this area include the development of introspective and reflective systems, contextual reflection, efficient run-time analysis and detection of system properties, and support for autonomous system evolution and model-centric development. Many of the ideas of
this section are described in more detail in [NDG+08].

3.2.1 Reflective Languages and Platforms

To support dynamic adaptation and evolution the behaviour of the running program has to be changed while it is executing. It seems plausible, that the ability of a program to reason about its own behaviour can greatly enhance the quality of the change process. This can be achieved by support for reflection: introspection provides the program with facilities to inspect its own structure and run-time environment, intercession enables the program to modify these elements. Current reflective systems possess some difficulties: it is often difficult to exactly specify the functionality or the part of a program that should be affected by a reflective change, and the performance of reflective systems is often order of magnitudes slower than that of a non-reflective implementation. Future research is needed to address these problems.

3.2.2 Contextual Reflection

One possible approach to limit the scope of reflective changes is to introduce the notion of context into the programming language. Then changes can be isolated by restricting them to certain contexts. Contexts are also a possible solution to the problem of inconsistency that can arise when updating large systems: as described in the previous subsection such updates cannot be instantaneous but they spread through the system according to a probabilistic space-time cone. By restricting changes to contexts in which they are valid the designer can ensure that the local view of each node remains consistent in spite of non-deterministic propagation of changes through the system. Future research can address adequate notions of contexts and ways to efficiently integrate them into programming languages and run-time environments.

3.2.3 Run-Time Infrastructure

With dynamically evolving systems, many properties can no longer be statically determined, instead the system will have to monitor its execution and deduce the presence or absence of properties based on run-time information. The development of efficient run-time infrastructures for these monitoring tasks, the detection of emergent properties from this run-time data and the connection of static analysis with run-time monitoring all represent challenging future research areas.

3.2.4 Support for Autonomous Adaptation

As detailed in Sec 2.2 heterogeneity can lead to situations where a node has to act differently, based on its environment or resource constraints. This gives rise to an interesting problem for autonomous adaptation: the models for these adaptation processes are not confined to a single abstraction layer in the system. Instead, changes to the network connectivity of a partner node may cause a re-evaluation of a node’s overall goals or strategy, and vice versa. This requires a seamless integration of different system models both inside and across abstraction layers.

Current approaches to model-driven development regard the model as a design-time structure. To enable autonomous adaptation and evolution of ensembles it will be necessary to provide access to the model at run-time and to causally connect the run-time model to the system’s execution. This approach has been called model-centric development. In this approach the programmer no longer operates on a textual or graphical description separate from the run-time model, instead design tools as well as autonomous adaptation of the system both operate on the same run-time model of the system.

\textsuperscript{2}The possibility to reason about the program’s behaviour is not strictly necessary. Some approaches, e.g., programs generated by genetic algorithms, rely on a combination of fixed structural properties (the reproduction mechanism) and a fitness function to “evolve” programs with desired properties from an initial population of programs. However, to achieve goal-directed modifications of programs some reflective capabilities seem desirable.
Proposed research areas for autonomous adaptation concern the causal connection between models and run-time behaviour, the autonomous evolution of run-time models, and integration of inter- and intra-layer abstractions.

3.3 Methods, Tools, Processes

To support the ensemble engineer throughout the development process, research in the areas of formal methods for specifying and analysing system properties, development tools, and development processes is necessary.

3.3.1 Methods

Formal methods for specifying, reasoning about, and analysing ensembles will represent an important component in the design and development of dependable ensembles. The current state of the art in formal methods is, however, not advanced enough to be useful for the development of ensembles.

One of the challenges for formal methods is to scale to the size of realistic ensembles. Tools such as model checkers often suffer from state-explosion problems when faced with system consisting of large numbers of nodes. In many cases these problems can be overcome by finding appropriate abstraction techniques to reduce the state space, or by using novel methods that do not suffer from state explosion. For example, the Performance Evaluation Process Algebra (PEPA) can not only be analysed by using continuous-time Markov chains, it also offers a new analysis technique based on sets of coupled ordinary differential equation that offers significantly better scalability for systems with large numbers of repeated components. A further problem for formal methods is that we often lack compositional descriptions of systems. Since compositionality is often a prerequisite for the scalability of formal methods, this limits formal analysis methods to small systems. Finding novel descriptions that allow us to present systems which currently appear to be non-compositional in a compositional manner is therefore an interesting research challenge.

Many current formal methods are only usable for closed systems since they rely on a closed world assumption, and since analysis results are not robust in the face of minor changes to the system, i.e., even minor changes to the system structure require the analysis to be re-run from the beginning. Novel methods that can be used to describe and analyse open and dynamic systems, and that scale to the size of realistic ensembles are an important research goal in the area of formal methods.

Another research area is finding replacements for the notion of “correctness according to a specification” which are more appropriate for ensembles. In general it will not be possible to specify the behaviour of an ensemble in the amount of detail that is necessary to apply formal methods due to the size of the ensemble and the complexity of its interactions and behaviours. Instead the notion of correctness will have to be replaced with notions of “fitness for a purpose” that allow the expression of important system characteristics in a formal, yet concise way.

More detailed information about the use of formal methods for ensemble engineering can be found in part II of [WBHR08].

3.3.2 Tools

While many concepts from advanced development environments have found their way into wide-spread use, the research frontier in software development environments (SDEs) has advanced surprisingly little in the last decades. Many of the concepts and ideas present in current software development environments can already be found in the state-of-the-art development tools from two decades ago. For future generations of tools we need a conceptual change in the role of development tools: it will no longer be sufficient for modelling tools to be essentially disjoint from the rest of the development process, with code generation and limited round-trip engineering as the only connection between models and code. Instead future SDEs need to provide the developer with support on a much higher level than currently
possible, e.g., by understanding the high-level models of the system and by explaining errors in terms of the models and not simply as backtraces; by integrating powerful static analysis and dynamic monitoring techniques for massively parallel systems and tools for aggregating and visualising the system behaviour, or by supporting the uncoordinated parallel development processes that will be the norm for many kinds of ensembles.

3.3.3 Development Processes

Current development processes clearly separate design and development form deployment and operation of a system, and they typically assume that the development team has control of the whole system, or at least the system within a well-defined boundary with specified interfaces to the outside. For many ensembles these assumptions will no longer hold.

In service-oriented systems we are already seeing a reliance on services not under the control of the system developers, in some future systems, e.g., societal ensembles, these dependencies and interdependences will increase significantly. Therefore, development will proceed in a much more parallel manner, with many of the concurrent development tasks being performed by teams which may not even be aware of each other. Future research will have to identify development processes that can help developers to build reliable systems even under the challenging conditions of uncoordinated parallel development.

3.4 Testbeds and Systems

To validate the ideas developed in the previous sections, they have to be applied to realistic case studies. WG1 proposes testbeds based on two particular important kinds of ensembles which differ significantly in their typical characteristics: physical and societal ensembles.

3.4.1 Physical Ensembles

Figure 4: A fractionated spacecraft is an example of a Physical Ensemble. The image is an illustration from DARPA’s F6 Program to design next-generation satellites. Contrary to traditional monolithic approaches, the functionality of a fractionated satellite is distributed across several heterogeneous modules that communicate wirelessly.
Physical ensembles are ensembles which contain many nodes that sense or effect the physical environment. Physical ensembles are closely related to cyber-physical systems, although the latter term is often defined in a more comprehensive manner that also encompasses embedded systems that are too small to qualify as physical ensembles.

Most groups of modular robots, sensor networks, distributed control systems for industrial plants, programmable matter, and many ad-hoc networks are examples for physical ensembles. In contrast, grid computing environments are not examples for physical ensembles since they lack the connection to the physical environment and the variable network topology; many traditional embedded systems are not physical ensembles because they are either not connected to a network, or only connected to a central server. Additional information about engineering physical ensembles can be found in [Tal08, SM08].

Because of their importance for the industry and the large number of practical applications, physical ensembles are an important type of ensemble. Several notions that have already been mentioned in the discussion of general ensembles take on a particularly decisive role in the context of physical ensembles: locality and physical situation are crucial for the correct operation of sensors and actuators; resource-constraints are often more severe and the restriction to nearest neighbour communication particularly frequent, because physical and environmental considerations limit the availability of computational power and communication bandwidth in many nodes. For example, battery-powered sensors in a sensor network are often severely restricted in their transmission power in order to conserve battery capacity. For many physical ensembles, predictability and high confidence are crucially important properties that need to be preserved even in the face of partial failures and system reconfigurations.

There are many different varieties of physical ensembles; we propose to concentrate the short to mid-term research on two particular kinds of physical ensemble: massively distributed physical systems and the Internet of things.

Massively Distributed Physical Systems. Massively distributed physical systems (MDPS) are a kind of physical ensemble that might offer particularly interesting research results: MDPS are massively parallel, low-order homogeneous physical ensembles where all nodes work toward a common goal, i.e., MDPS are systems containing massive numbers of possibly unreliable, resource-constrained computational units interacting with the physical world, with only a low number of different component types, where nodes might be competing for shared resources, but where all nodes share the same overall goal.

We consider MDPS to be good objects for research on ensembles, since they have several properties that simplify their design: while nodes compete for shared resources, they are not adversarial; there is a limited number of different node types which simplifies the prototypical construction of these systems, and MDPS can often be designed with a relatively simple network structure. Research problems for which MDPS present particularly interesting challenges are, among others, explicit spatial reasoning and shape control, and the assembly and architectural dynamics of large-scale structured ensembles.

Networks of Things. With the decreasing size and cost of microprocessors and wireless networks we are seeing a trend towards computational devices that no longer resemble traditional computers. Instead everyday items such as portable phones, music players, household appliances, or cars are becoming network aware. In addition, new special purpose interaction devices are being developed. For example, siftables are “cookie-sized computers with motion sensing, neighbour detection, graphical display, and wireless communication.” Collections of siftables allow new user interaction paradigms to be developed since the user can control system behaviours, e.g., by grouping siftables that represent different system features together. Future research should investigate the challenges and opportunities presented by interacting with the system via these new devices.

3.4.2 Societal Ensembles

Societal ensembles are ensembles that are closely connected to humans, e.g., smart cities, ambient assistance, or global virtual enterprises. In contrast to the massively distributed physical systems described
in the previous section, societal ensembles will generally contain large numbers of different components, with different and often competing goals, and with often very complex and unstructured interaction patterns between heterogeneous parties. Additional information about related topics can be found in [Fia08, Hig08].

Research in this area will have many connections to research on cognitive systems and ubiquitous computing. One major concern will be to investigate the social and emotional perception of human activities, the dynamics of purposive interactions, and how the structure of evolving societal interactions can be reflected in the architecture and the design of the software. The evolution of societal ensembles has to be a long-term process that goes beyond single-run adaptation, and may span years or even decades. Societal ensembles have to support diversity and context awareness: adaptations between instances of the same component type may vary widely, based on the environment and history in which a node of the system is deployed; these adaptations have to be preserved even when the software on the node, or the whole node, is updated. Individual nodes, as well as the whole system, have to maintain social coherence and perform according to the social expectations of users even while they are adapting to their environment.

Privacy, identity management, confidentiality, security and trustworthiness are all particularly important for societal ensembles, since these ensembles will process large amounts of personal data of their users. One particularly important research area is the combination of these issues with the social context, e.g., a social ensemble may divulge private medical data to the doctor treating a patient, but it may not do so over a loudspeaker if people who are not authorised to receive this information can overhear the spoken information.

Societal ensembles will be a particularly interesting test bed for research on heterogeneity and on ensembles where individual nodes exhibit partial cooperation and partial competition. Heterogeneity is a result of the many different types of devices that humans regularly use, and of the individual preferences for different kinds of devices: whereas the designer of a physical ensemble may standardise, e.g., on a single model of sensor node, a similar standardisation on one kind of mobile phone or me-
dia player is not feasible for societal ensembles. Different actors in social systems, whether they are humans, computers, or companies, have partially competing goals, which lead to intricate patterns of cooperation and competition. For example, it is customary for companies to cooperate as partners in one project while simultaneously competing for another project. This complex pattern of simultaneous competition and cooperation leads to other effects, e.g., partial sharing of information while keeping secret other information which provides a competitive advantage. Recognising and appropriately dealing with these complex social situations and behaviours will be another interesting research direction for societal ensembles.

It is evident that many future ensembles, but in particular societal ensembles, will have to deal with larger and larger amounts of data. This data is increasingly being acquired from heterogeneous and federated data sources, containing potentially contradicting or incomplete information, and often in semi-structured formats. The reliability and trustworthiness of different data sources will vary widely. Processing these kinds of data will remain one of the challenging research areas in ensemble engineering.

4 Cross-Thematic Group Synergies

The research proposed by WG1 is closely related to the research areas of the other InterLink working groups: on the one hand, most ambient computing environments as well as most future intelligent and cognitive systems will be ensembles, and thereby ensemble engineering will be a foundation for building these kinds of systems. On the other hand, many ensembles, e.g., most of the societal ensembles described in Sec. 3.4.2, contain nodes which are themselves intelligent or cognitive systems, or even ambient computing environments. For example, a smart city may contain a large number of rooms equipped with ambient computing environments; a system for assisted living may contain systems that sense the mood or cognitive state of the inhabitants.

Another interesting synergy is the use of techniques from ambient computing environments, and from intelligent and cognitive systems in the development of ensembles: as described in Sec. 3.3 the development process for many ensembles will be highly parallel, often with groups working for different organisations developing interacting parts of the system in parallel. How can we use intelligent and cognitive systems to support developers in these complex environments? Can ambient computing environments be used to increase the cooperation between geographically and organisationally diverse groups of developers, and to improve the interaction in more closely cooperating groups?

5 Suggested Instruments for International Cooperation

We see three major possibilities for international cooperation:

1. International research projects, funding researchers and PhD students as well as travel, etc. The precedent for this is the cooperation with Israel and Switzerland which participate financially in EU projects in order to avoid double reviewing. A prerequisite for the success of that kind of programme is that an internationally accepted legal framework for intellectual property rights (IPR) of participants from different countries and continents can be put in place.

2. Coordinated Actions for organising workshops, summer schools, training and newsletters. This could take either the form of a horizontal support action to strengthen international cooperation and possibly fund travel, or it could consist of the submission of a Coordinated Action to the EC and a proposal to the NSF call “Partnerships for international research and education” to coordinate the global community of researchers in the area of ensembles that is arising in the EU and the USA.

3. Bi-lateral agreements for funding travel. Such a process exists for cooperation between Australia and Europe. The action in this case is to propose a Coordinated Action for ensemble engineering.
in order to prepare projects and start building a research group. The goal would be to define or refine a research agenda in the short term, to build critical mass for a call.

6 Conclusions

It is hard to over-estimate the influence that software-intensive systems already have on economic and social developments throughout Europe and the world. Given the advantages of software to control systems and the decreasing cost of controllers, the importance of software-intensive systems will still increase in future years. The European Union has already started important research projects in this area, but continuing future research efforts are needed for Europe to stay competitive and to ensure that the needs of the European society are met by future developments. The Interlink WG1 is convinced that ensembles will become the most important area for new research and future development: software-intensive systems with massive number of nodes, nodes with sophisticated behaviour, or complex interactions between nodes, operating in open and non-deterministic environments, with variable network topology and a need for adaptation to new requirements. The trend to these kinds of systems is already evident in software-intensive systems which are currently deployed or being developed. However, significant research efforts are needed to address the inherent complexity of ensembles, to ensure the trustworthiness, security and reliability of future ensemble, and to ensure that they can be engineered cost effectively.

The scope of the research tasks and their importance for the future of the European and international society are compelling reasons for the research to transcend national boundaries and for research projects to be undertaken on a European level. To further accelerate the research efforts, to share knowledge and expertise with non-European countries, and to distribute the significant financial burden of the necessary research, international cooperation beyond the boundaries of the European Union would be advisable.

A List of Contributors

- Gul Agha, University of Illinois at Urbana-Champaign, USA
- Jean-Pierre Banâtre, University of Rennes, France; INRIA, France
- Gabriel Ciobanu, A. I. Cuza University of Iași, Romania
- Pierre Cointe, Ecole des Mines de Nantes, France
- William Cook, University of Texas at Austin, USA
- José Fiadeiro, University of Leicester, UK
- Pascal Fradet, INRIA, France
- Jean-Louis Giavitto, University of Evry, France
- Fausto Giunchiglia, University of Trento, Italy
- Seth Goldstein, Carnegie Mellon University, USA
- Manuel Hermenegildo, Universidad Politécnica de Madrid, Spain; University of New Mexico, USA; IMDEA Software, Spain
- Teruo Higashino, Osaka University, Japan
- Matthias Hölzle, LMU München, Germany
- Stephan Jähnichen, TU Berlin, Germany; Fraunhofer FIRST, Germany
- Michael Johnson, Macquarie University, Australia
- Hélène Kirchner, INRIA, France
- Insup Lee, University of Pennsylvania, USA
- Zhiming Liu, United Nations University, Macao, SAR China
- Vicenzo Manca, University of Verona, Italy
- José Meseguer, University of Illinois at Urbana-Champaign, USA
- Jayadev Misra, University of Texas at Austin
- Oscar Nierstrasz, University of Texas at Austin
- Axel Rauschmayer, LMU München, Germany
References


[BFR08] Jean-Pierre Banètre, Pascal Fradet, and Yann Radenac. The chemical reaction model—recent developments and prospects. In Wirsing et al. [WBHR08].


[Fia08] José Luiz Fiadeiro. On the challenge of engineering socie-technical systems. In Wirsing et al. [WBHR08].


[Hig08] Teruo Higashino. Design and deployment of large-scale software-intensive systems in urban districts. In Wirsing et al. [WBHR08].


[HWR08] Matthias Hölzl, Martin Wirsing, and Axel Rauschmayer. Software engineering for ensembles. In Wirsing et al. [WBHR08].


[NDG+08] Oscar Nierstrasz, Marcus Denker, Tudor Gîrba, Adrian Lienhard, and David Röthlisberger. Change-enabled software systems. In Wirsing et al. [WBHR08].


[SM08] Lui Sha and José Meseguer. Design of complex cyber physical systems with formalized architectural patterns. In Wirsing et al. [WBHR08].


[Tal08] Carolyn Talcott. Cyber-physical systems and events. In Wirsing et al. [WBHR08].

