ABSTRACT

Ultra-Large-Scale Systems are considered to be the new system generation. For these systems current development approaches are not well suited, since they do not scale well for large systems. One view onto such systems is to see them as IT Ecosystems comparable to complex ecosystems from nature and biology. We will present an example for such a system, the smart city. Furthermore, we will present challenges that arise from engineering and operating IT Ecosystems.

Categories and Subject Descriptors
D.2.11 [Software Engineering]: Software Architectures; K.4.3 [Computers and Society]: Organizational Impacts

General Terms
Design, Human Factors

Keywords
IT Ecosystems, Ultra-large-scale Systems, Challenges, Chaos

1. INTRODUCTION

In our days, the trend towards “everything, every time, everywhere” is becoming more and more obvious: the Internet, for example, can now be accessed using mobile phones. Furthermore, electronic assistants, so-called “information appliances”, like network-enabled PDAs, WAP-capable cell phones and electronic books or tourist guides, are available in stores all over the country.

The continuing progress of all IT domains towards “smaller, cheaper, more powerful” is the main driver that enables this trend. Moreover, new developments in the field of materials science like midget sensors, organic light-emitting devices or electronic ink, and the evolution in communications technology, especially in the wireless sector, contribute to embedding active software components in nearly every commonplace and industrial object, like for instance smartphones, refrigerators, and cleaning machines. These devices have processing and communication capabilities and thereby provide the execution hosts for the corresponding active software components.

Those active components are more and more used within an organically grown, inhomogeneous, and dynamic environment. End-users of these devices want them to communicate amongst each other in order to provide additional functionality, which they cannot provide on their own. For example a user wants his fridge to create a shopping list and send it to his PDA, which will then guide him to the shops. These shops are selected based on an internet price comparison taking the current position of the user into consideration.

Therefore the devices and the executed software components have to be connected with software services provided by various software systems of the ambient environment. Such an organically grown system with emergent functionality represents an Ultra-Large-Scale Software-Intensive System (ULSSIS) [2]. This new dimension of software systems will be the predominating form of systems in our daily environment.

One can imagine, that ULSSIS cannot be specified and developed as a whole, due to many reasons like for instance high degree of complexity, different life-cycle of the various components, and diverse vendors of the parts of such a system. In analogy to architecture we are able to plan and build buildings but we are hardly able to plan and build whole cities from scratch.

In the remaining paper we will provide an sample for a ULSSIS - the Smart City. Based on this demonstrative example we will motivate and derive our metaphorical roadmap “From Ecosystems to IT Ecosystems” to cope with the engineering challenges for ULSSIS. Similar to natural ecosystems, IT Ecosystems are based on a set of fundamental rules to balance individual autonomy and overall control. Finally we present the main research issues associated with IT Ecosystems for the next years.
2. THE SMART CITY

In southeast Asia emerging cities like New Songdo City are designed as so called smart cities from the beginning [6]. These smart cities are characterized by the ubiquity of IT systems. These IT systems are available to the citizens at every place, taking the context of them (like location or their current activity) into consideration. The systems belong to different domains like smart-living systems supporting elderly people in their daily life [5], smart-working systems supporting the workflow in a company, or smart-transportation systems supporting the public transportation within the city.

These systems are organized in a decentralized way and contain autonomous, mobile IT components deployed on intelligent clothing, smartphones or cars. Moreover they are self-organizing and exist in parallel. In order to assure the whole functionality of a smart city, we have to balance the forces of the individuals within this huge IT Ecosystem.

Smart cities are seen as a prime example for IT systems of the future regarding their complexity and autonomy. Therefore they are a good application example motivating the necessity of new approaches for specification, design and validation.

Our approach is to change the viewpoint on these systems. Instead of treating them as a whole together with their environment, we need to treat them as part of a larger IT Ecosystem. In analogy to biology’s ecosystems, these IT Ecosystems share the following characteristics:

- They consist of humans, software-intensive systems, and their environment.

- They are a continuously changing community of individuals, which may compete against each other for resources or cooperate with each other in order to reach their individual goals (autonomy).

- They include methods for controlling and monitoring their state of function, like measuring methods or principles to incentive their individuals. By these means, the survival of the system is guaranteed (control).

Thus, an IT Ecosystem can only function properly and be stable, if autonomy and control are carefully balanced.

In the following we will introduce several challenges associated with IT Ecosystems and sketch first means, how to overcome them.

3. CHALLENGES AND APPROACHES FOR IT ECOSYSTEMS

A core aspect, which needs to be addressed in order to achieve IT Ecosystems is the autonomy of all participating IT components. These IT components are called individuals in the following. When autonomous individuals interact in an IT Ecosystem the question comes up, how the IT Ecosystem can be controlled.

From biology we learned, that an ecosystem is managed by a set of basic rules. These rules ensure, that the ecosystem is striving for an equilibrium of forces for all individuals. Otherwise the ecosystem will break down. You can see an imbalance in the forces of the natural ecosystem for example during algal infestation leading to massive fish mortality. Similar to ecosystems, IT ecosystems should strive for balanced states that guarantee correct functionality of the system.

For large systems of both kinds, ecosystems and IT Ecosystems, the sets of basic rules are not fully understood. In the case of IT Ecosystems, those rules have to be followed during the design of parts of the IT Ecosystems, and enforced during its operation. In the following, we investigate the impacts of the interpretation of ULSSIS as IT Ecosystems on two facets of engineering such systems. These facets are amongst others addressed by Braunschweig University of Technology, Clausthal University of Technology, and University of Hannover. These three universities are currently on the way to establish a common graduate school for IT Ecosystems to focus on this research challenge.

3.1 Requirements and Architecture Modelling for IT Ecosystems

It is well known that requirements engineering and architectural design are key factors for the success of software and that these early phases are intertwined [3]. Requirements can heavily influence the choice of architectural solutions – consider, for example, the requirement for fast data input for an information system realized by an ultra-thin client, which would be hard to achieve (cf. [4]).

The architectural design of software intensive systems is mostly done at the level of components. Current component-based software development (CBSD) approaches rather focus on the design of software systems than on the system’s high-level, coarse-grained architecture [7]. They provide modeling techniques to describe the concrete structure of components and their functionality, how they are connected and how they interact. Many of them use UML-like notions or textual specifications to abstract from a certain component implementation technology or a dedicated programming language. In most cases, this results in a very small step of semantic abstraction in specifying the system but more in a “syntactic” abstraction from implementation [1]. As an effect, they are not appropriate to explicitly model the fundamental rules and principles of a software architecture.

Software engineers mostly follow those fundamental rules, expressed as coarse-grained patterns, reference architectures or architectural guidelines rather by intuition and implicitly, than by explicitly modelling the architecture at that level of abstraction. This is critical in the case of IT Ecosystems, since those fundamental architectural rules describe allowed structures and interactions between components, and therefore are clearly required to balance their autonomy and controllability.

Thus, it is necessary to express those fundamental architectural rules for IT Ecosystems explicitly for different reasons:

- High-level requirements influence the set of architectural rules on a level above of components. Reference architectures like Three-Layer-Information-Systems abstract from the concrete functionality of the system and components. Without modelling those rules explicitly, it is not possible to ensure traceability at that level.

- The fundamental architectural rules constrain the design of parts of the IT Ecosystems and components. By explicitly modelling them, they can be used to automatically enforce them during design.
Those architectural rules should be checkable at runtime due to the dynamic and adaptive character of IT Ecosystems. Therefore, they have to be modelled.

The first aspect emphasizes the need for an engineering approach, which ensures that the selection of architectural rules for an IT Ecosystem is based upon the requirements which stakeholders of the system may have. Thus, an architectural rule has to contain information about quality properties that can be assured if it is applied.

In the NTH School for IT Ecosystems we therefore are going to take an approach based upon architectural rules and goal-oriented requirements engineering [4]. Goal models are analyzed for structures which separately formulate quality properties that are influenced by the architecture (architectural requirements). Architectural rules are complemented by architectural assurances, which define which quality properties can be assured. The main challenge here is to consider the composability of the architecture and its rules for the evaluation of composite architectural assurances.

3.2 Runtime Support for IT Ecosystems

Next to the modelling support, we need a runtime environment for IT Ecosystems to control their structure and behavior. On the one hand, this runtime environment can be used to monitor the rules introduced in Sec. 3.1. On the other hand, additional rules controlling the wiring of IT Ecosystems can be executed by the runtime environment. These rules include rules guaranteeing the correctness of the IT Ecosystems or rules considering context information during wiring as described in more detail in the following.

3.2.1 Support for dependable IT Ecosystems

Users of an IT Ecosystem expect its individuals to collaborate autonomously with each other and provide a real added value to him. On the other hand, he depends more and more on the resulting IT Ecosystem. Hence its correctness has to be guaranteed even though it has never been developed and tested as a whole in advance. IT Ecosystems need to have the ability to dynamically attach and detach dynamic adaptive IT components during runtime. Moreover they need to detect and avoid possible resulting incorrect system configurations during runtime.

It is obvious, that an IT Ecosystem integrates individuals from different vendors. The integration of these individuals may be an operation performed by a system administrator or they may be seamlessly integrated without further user interaction. The support of seamless integration is indispensable for IT Ecosystems, since they are supposed to integrate mobile devices like smartphones, which may enter or leave a system permanently.

If we wire a system out of individuals at runtime, we will have to deal with the dependability of the newly established system. Due to the explosion in combination of these components and due to the fact, that IT Ecosystems are composed of components from different vendors and are not developed as a whole, we can not verify the correctness for all possible resulting systems in advance. Instead we have to provide a mechanism, which enables an improvement of dependability compared with the situation of stupefying composing the system of syntactically matching components without taking any dependability actions.

Our approach is applying runtime testing to ensure, that the components appearing in IT Ecosystems collaborate correctly. Components requiring interfaces from external components therefore need to specify test cases for their required interfaces. The runtime environment will execute these test cases on external components before wiring them together with the component user. Therefore incorrect system configurations can be detected in advance.

3.2.2 Support for context-adaptive IT Ecosystems

After eliminating incorrect system configurations of an IT Ecosystem as described before, several different correct system configurations may remain. The runtime environment needs a mechanism enabling the selection of the “best” configuration in this case. One approach to eliminate further system configurations and therefore find the best one in the current situation is to take context information into account.

Context information in general can be seen as any information which is able to describe the situation of individuals in terms of location, time, and identity.

Two aspects play an important role to achieve the goal of finding the best configuration considering context information. First, each individual has to define what it is able to provide to the other individuals which in turn may be dependent on context information. At this point context information can influence the provided functionality.

Secondly, a system has to define what is wants to ensure to its environment while taking into account context information. Each single individual only can define what is the best for itself without taking into account interests of other individuals or the goal of the collective. Therefore, there have to be rules defining what the system aims at and how this can be reached by letting all individuals work together. One ant for example only knows where to go, not that all ants together want to enlarge their ant-hill.

The system rules and the information each individual provides about what it can provide under specific circumstances together with context information enables a system to find a valid configuration for the current situation, not only a correct one in terms of interface compatibility. For all of these aspects there is a need for appropriate description techniques together with an execution environment enabling the processing of this information. While there are available many approaches for describing context information and some approaches for describing system rules, there is still a lack in integrating these to approaches in order to realize ULSSIS.

4. CONCLUSION AND FURTHER WORK

The metaphor of an ULSSIS as a biological concept seems to be useful to comprehend the interactions and interdependencies in such complex and dynamic IT systems and applications. Our main task is to understand the rules to balance the two forces autonomy and control in an IT Ecosystem and to develop software system engineering approaches to achieve and retain it. We motivated this view by an application scenario - the Smart City. We postulate that due to complexity, life-cycle, and globalization issues classical engineering approaches are no longer applicable. IT Ecosystems cannot be planned, designed and implemented as a whole. Hence new software system engineering paradigms have to be developed.

It has to be investigated how such IT Ecosystems can be designed properly especially focusing on detecting, analyzing, and modelling the fundamental architectural rules of such systems. A dynamic adaptive infrastructure for IT...
Ecosystems is needed to integrate components into these systems. This infrastructure needs to be capable of ensuring the correct behavior of the newly wired system. Moreover it needs to consider context information during the wiring. Among others, these issues are tackled in the new NTH School for IT Ecosystems.

5. REFERENCES


