Chapter 1

Introduction

This introductory chapter aims to give an overview of the thesis context, starting with an introduction to the area of interest in Section 1.1, which focuses on the Future Internet and software service engineering. This includes an introduction of the definitions of a service, a service-based application, a composite service, etc. The aim is showing the position of software engineering in the Future Internet. A presentation of the motivations for the thesis follows in Section 1.2, and the main contributions of the thesis to the field of software service engineering are presented briefly in Section 1.3. At the end of this chapter, in Section 1.4, the structure of the thesis is outlined.

1.1 An Introduction to the Area of Interest: *the Future Internet*

It is undeniable that the Internet is becoming an important infrastructure for the growth of today's economy and society, where the infrastructure itself is also gradually becoming larger. Now, information technology is spreading into all areas of daily life, leading to an increasing amount of information and applications. New devices and technologies (e.g., protocol layers) are continually added, leading to an increasing complexity of software systems. Surprisingly, today's Internet that was designed in 1970s is mainly built only for information sharing. There is a mismatch between the original design goals and the current and future utilization of the Internet technology. A new system is needed, and we call this system the Future Internet [78].

In the Future Internet, Internet resources (people, media, services, devices, and networks) will be converged in terms of connectivity. Various autonomous comput-

ing and networked devices, from small (mobile devices, embedded systems, etc.) to powerful devices (desktops and servers) may be easily connected to the Internet network, in a plug-and-play manner. These devices can communicate and cooperate with each other to form a specific composite system or application. So, the cooperation is not only between people and devices, but also between devices. This is supported by the fact that as on-going miniaturization allows small devices such as sensors and actuators, to become autonomous, smart and powerful data-processing engines of all kinds are emerging. In contrast, the more devices are connected to the Internet the more applications are going to be diverged, leading to an increasing complexity of software systems and their development.

1.1.1 The Concepts of the Future Internet

To illustrate the Future Internet, several terms and related concepts have been introduced. In this section, three concepts are presented: the Internet of Things [100], the Internet of Services [95] and the Cloud Computing [109]. We will shows how the concepts are related to each other.

The first important concept illustrating the Future Internet is the Internet of Things [100]. Different perspectives may have different definitions of the ***things*** in the Internet of Things. However, in the context of the Internet technology, the thing can be defined as a physical or virtual entity that exists in space and time, and is capable of being identified and utilized according to its properties. Among important properties are the ID, location, name and behavior.

In [100] it is stated that:

> *The Internet of Things allows people and things to be connected anytime, anyplace, with anything and anyone, ideally using any path/network and any service.*

The definition above implies handling of several important issues. Two examples of the issues are how the connection between entities can be made seamlessly, and which addressing mechanisms should be used. Obviously, in order to be able to communicate with each other, each device requires a different address, an ID and a name that must be unique.

Referring to the object-oriented perspective, the Internet of Things can also be considered as the Internet of objects. The objects would be equipped with sensors enabling them to detect the environment and to adapt if the environment is changed. In a smart home environment [81, 22, 104] for example, all objects (devices) are

considered to be equipped with minuscule identifying devices and they are connected to the network. They are a kind of autonomous and smart objects that are able to do self-adaptation to the network. It will be possible to develop combined applications based on individual object's behaviors.

The objects in the Internet of Things provide or/and use functionalities. We can call the object's functionalities services, which in turn, introduces the term Internet of Services [95]. The vision of the Internet of Services is that everything is available on the Internet as a service, such as the software itself, the tools to develop the software, and the platform (operating systems, hardware and networks) to run the software. This is the main characteristic of the concept of the Internet of Services.

Another important concept of illustrating the Future Internet is Cloud computing [109]. The concept is a relatively new model of Internet-based computing, where resources in the Internet, e.g., servers, storage, networking, software, are provided as services. Cloud computing can be defined as an architecture where IT resources (hardware and software systems) are delivered to the users on demand without strict dedicated associations between customers and resources [109]. The delivery model for the resources is based on the utility using a pay-per-use pricing model. The key is that any kind of user can access the services, even inexperienced users. The main feature of all these service offering models is that they break up the previously monolithic ownership and control of the resources in the various technology layers and distribute them across multiple entities.

The concept of Cloud computing has a similarity with the concept of the Internet of Services, in which everything is provided as a service. Based on target service users, services in Cloud computing can be roughly divided into three main categories: infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS).

In the IaaS category, machines are virtualized and provided as services. These services include computational resources such as hardware and storage services including services for managing groups of these resources. The PaaS service category offers development platforms for various application domains. In the SaaS category, software applications are provided as services. The users of these services range from private users of services to enterprise customers operating entire business processes.

The Cloud computing concept has also a similarity with the concept of the Internet of Things. Cloud computing uses an implicit assumption that resources are of a type of entities that is able to host and process data. However, this is the main issue in the Internet of Things. In the Internet of Things, every device is a networked,

smart and self-adapted resource. The devices also host and process data.

A common similarity of the three concepts mentioned above is about functionality (i.e., service) that is provided by an entity and can be used by others. The literature provides many informal definitions of a service that were inspired mainly by applications in the telecommunications domain. However, generally, a service can be defined as functionality offered to a service user by a service provider. Both service users and service providers can be human beings, enterprises, as well as software and hardware entities (i.e. programs and devices). In the context of the Internet of Things, services are software systems that are embedded on devices. However, it must be noted that services are not only hosted on small and embedded devices, but could also be hosted by more powerful devices, physically or virtually. With this situation, the Internet will contain a huge amount of embedded services.

The problem is that there is no standard of formal method and language for implementing services. The services may have been implemented in different programming languages. They can also run on different unrelated environments. This leads to a problem when the service users want to use the services in a composite manner. To solve the problem, a service broker in terms of service-oriented architecture (SOA) [28] can be considered. However, SOA is only a concept. We need tools and methods for implementing the concept. A rapid, flexible, high abstraction level and efficient tools for the development software service is required.

1.1.2 The Software Engineering Perspective

The classic issue of software development concerns the development methodology and languages (i.e. programming and modeling languages). There is still criticism of software systems development languages and the methods employed, such as high cost, long time-to-market, and poor flexibility. Software reuse is one of the proposed solutions to the criticized development methodology as it promotes reductions in cost and time-to-market. In this solution, a new software system can be promoted (created) by means of collaboration of existing software units. An example of this solution is the use of software components (i.e., software unit) instead of manually handwritten code from scratch to develop a software system, that was proposed by McIlroy in [57]. Based on his idea, several models of a software unit were introduced. Modules, objects, components are examples of models of a software unit.

However, the demand for software to live in an open world and to evolve continuously as the world evolves, is leading to the evolution of software methodologies and technologies. The evolution can be seen as a progressive journey from rigid

to flexible, static to dynamic, centralized to distributed solutions [25]. The history of software engineering shows a progressive departure from the strict boundaries of the closed-world assumption toward more flexibility to support continuous evolution [4]. Over the past years a major step of evolution in this direction has been made possible by the birth of the concept of a service.

With regard to the software component models introduced in [57], we consider that a service is only another type of model of a software unit. In this context, a service has evolved considerably from the older models: modules, objects, and components. They are only different in terms of abstractions, encapsulations and ownerships. Among these models, the service is the highest abstraction level and the most encapsulated model. Accordingly, depending on the models of software units, software systems can be categorized as module-oriented systems, object-oriented systems, component-oriented systems and service-oriented systems. The service-oriented system is also known as a service-based application.

Unfortunately, traditional software engineering methods and approaches are not fully appropriate for the development of service-based applications. In the context of service-oriented systems, these limitations have led to the emergence of software service engineering (SSE) as a new specialist discipline. However, research activities in this area are still immature, and many open issues remain. There is an urgent need for the research community and industry practitioners to develop comprehensive engineering principles, methodologies and tools support for the entire software development lifecycle of service-based applications [101].

Service engineering can be defined as the set of methods, techniques and tools to specify, design, implement, verify, and validate services that meet user needs and deploy and exploit these services, over current or future networks [111]. For the service development, service developers must be able to specify services so that they can be discovered and used and must be able to test these services in such a way that the tests reflect their complex and heterogeneous operating environment. According to [111], service engineering covers three important domains:

1. Service creation and tools: A software engineering platform.
2. Service management: The way a service is run and accessed by the users.
3. Network architecture: The way a service is delivered to the users.

Similarly, in [11], see Figure 1.1, the first domain is introduced as service engineering domain, the second domain is called service deployment and execution domain, and the third domain is called a service platform. The service engineering domain includes methods and tools (i.e., service creation environment, SCE) for

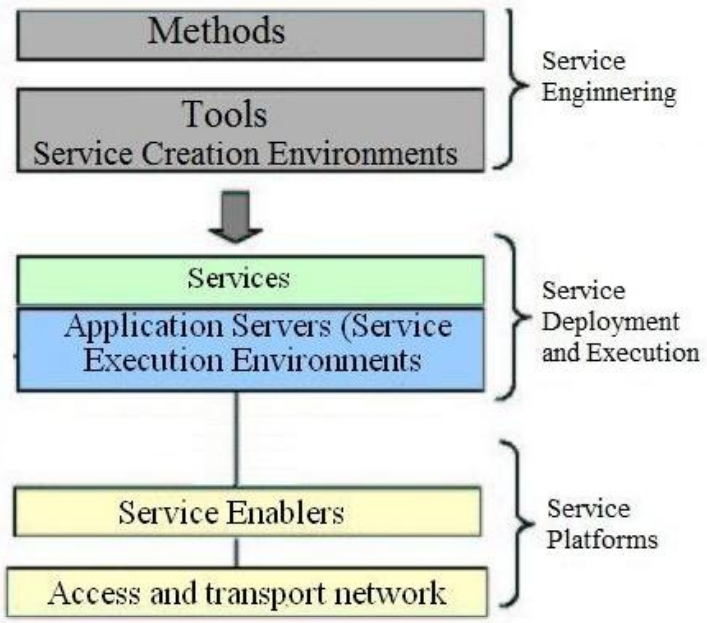

Figure 1.1: The Service Engineering Domain [11]

software services. This thesis addresses on the first domain of service engineering [111], that is service creation method and tools [[11]].

1.2 Motivation

Only with appropriate software will it be possible that the Future Internet comes to life as imagined. Within the context of the Internet of Services, creating service-based applications using the huge amount of services is a challenge. This includes the use of resources that are delivered in form of IaaS, PaaS and SaaS models in the context of Cloud computing. Within the European community [1], software development by composing pre-made services has been proposed as one of the research agendas during the years 2010-2015.

For the creation of service-based applications, different approaches and methods may be used. However, the approaches and methods should meet the three criteria below.

1. They must support for ***exploring the reusability*** of existing services.
2. The design processes should not depend on the target environment.
3. The service developers and service integrators should be shielded from complexity.

Two examples of approaches and methods are software composition approaches [3](was originally introduced in [57]) and model-driven development approaches (MDD) [71]. The first approach supports exploring the reusability of software components. In the last decade, the approach has been adopted in several of today's large-scale software projects in the area of distributed systems. With regard to the software component that was introduced in [57], we treat a software component as a software unit. For the composition of software units, software composition systems [3] can be applied.

The second approach uses models to specify, design, implement, verify, and deploy software systems. When doing model-driven development, software developers will work on model levels in a platform independent manner. The reason is that the MDD focuses on the separation of the problem domain from the implementation domain. This enables software developers to focus on the problem solutions on a high abstraction level rather than the implementation details. By doing this, they do not need to pay attentions to how the service models will be implemented considering the fact that the implementation is done automatically by machines (i.e., code will be generated automatically).

With regard to the complexity of software systems, the aims for both software composition [57] [3] and model-driven development (e.g. MDA [71]) are similar in which they are used for managing the complexity of software systems and their development. Having benefited from these approaches, the author of this thesis has been motivated to propose the use of MDD for the service creation in the Internet of Services. Employing this idea, the composition of services can be done using models at different abstraction levels, while the executable composite services can be generated automatically.

In fact, the use of models for the development of software applications has been done for a long time. For example, software developers often use high-level models (e.g. instance, box, and arrow sketches) to reason and communicate about a software system. The main problem with the use of high-level models is that they are almost always inaccurate when the design models would be implemented using specific programming languages manually [63]. This is a serious problem, since a model in MDD is used as a formal specification of a software system. A software system should be generated automatically from this formal specification, either by model interpretation or code generation.

Services in the Internet of Services may be hosted by small devices. Of the small devices, this thesis focuses on personalized and embedded devices which have different capabilities and possible configurations. For the development of such em-

bedded services, concrete and low-level information about the device capability and configuration is often required at the beginning of the development processes. In contrast, within the MDD context, such low-level information will be given later at the lowest level of design models, which is usually done at the end of the development processes. This is because the MDD development process goes from abstract (i.e., models) to concrete (i.e., implementations). If we want to apply MDD approaches for the development of embedded services, then we have to abstract the concrete information about the devices onto model levels. This abstraction process goes from concrete to abstract. So, there is a mismatch between the MDD approach and the software development approach for personalized and embedded devices.

The fundamental research question of this thesis is formulated as follows:

> *To what extent and how - can model-driven development approaches be applied for the creation of composite services and their deployment on personalized and embedded devices in the Internet of services?*

The problems and challenges of the use of MDD for the development of service-based applications will be further analyzed and presented in Chapter 2. Furthermore, the fundamental research question above will also be further decomposed, and analyzed and presented into six sub questions in Section 2.3.

1.3 Main Contributions

Model-driven development is not a new methodology for software engineering. However, the idea of applying the model-driven development methodology to service creation in the Internet of Services is relative new. This thesis focuses on service engineering as introduced in [11] which includes methods and service creation tools. More specifically, the thesis focuses on model-driven and compositional methods for service creation. The thesis proposes a method for composing services and for providing the composite functionality as a new service. The contributions of the thesis are listed as follows:

1. **A New Service Engineering Method for Service Creation in the Internet of Services.**
   The main problem of service development in the Internet of Services is that the *things* in the Internet of Things might implement services using different technologies and might also use different programming languages. A methodology to develop service-based applications using services that have

been implemented using different technologies is needed. Furthermore, when the service-based application has been developed, questions about how the applications can be provided as a new service and how the services will be deployed in different devices with different capabilities and various device configurations are interesting to be solved. The thesis contribution is a methodology of service engineering that is independent of technologies and supports for exploring the reusability of existing services. The thesis proposes a method called PMG-pro (present/abstract, model, generate and provide) to answer the fundamental research question.

The method has been presented in the paper "PMG-pro: A model-driven method for the development of service-based applications in a heterogeneous services environment" [92] and is published in the *Proceeding of IEEE International Conference on Software Engineering and Service Sciences, ICSESS 2010* and the *Proceeding of SDL Conference on Software 2011* [93]. A poster "Model-driven Approaches for Service-based Applications Development" [91] of the method was presented in the *Proceeding of 5th International Conference on Software and Data Technologies, ICSOFT 2010*.

2. **An Automated Service Presenter and Service Abstractor.**
Abstraction is an important key for the success of applying MDD. A service model is an abstract representation of a concrete service. Using abstract service models, new service-based applications can be modeled on model levels. Thus, we need a service presenter to present services, and a service abstractor to abstract services.

The presentation process involves model transformation methods [73, 8, 70]. The service presenter transforms a service description of a concrete service at run-time, to an abstract graphical service model. Different notations or symbols that conform to the chosen modeling languages can be used to present the services. The service graphical representation is an abstract service model. A service abstractor is used to construct more abstract service models in hierarchical service taxonomy.

The thesis contribution is an automated service presenter that is able to present graphically (embedded-) services from service descriptions and a service abstractor that constructs more abstract models. With the service presenter and service abstractor, it would be possible to present concrete services as abstract services. Using the abstract services, software developers can model new service-based applications using different embedded software systems

on different devices, on the model level. The contribution of the service presenter has been presented in the paper "Presenting Reusable Service Models in Model-driven Service Engineering" [88] and is published in the *Proceeding of International Conference on Future Information Technology, ICFIT 2010*.

A special service presenter has been developed for supporting end-user service compositions at run-time. The contribution has been presented in the paper "An Automated Services Presentation Method for Supporting End-user Compositions" [86] and is published in the *Proceeding of Conference on Information Technology and Electrical Engineering, Yogyakarta, Indonesia.*

3. **A new method of handling device capabilities and configurations.**
   The promise with the model-driven development approaches for the development of software applications is that new applications can be specified, designed, analyzed, and verified by models and thereby, code for specific platforms of target devices can be generated automatically from the models to have running applications. However, for small devices (mobile phones for example), device capabilities and configurations of the target devices are unknown at design-time. This introduces difficulties for the automated code generation. To be able to have a fully-automated code generation, we need to have knowledge of the underlying platforms.

   In addition the graphical service model, the service presenter in PMG-pro also generates source code conforming to a specific programming language. The source code implements the service invocations to the concrete service. We call these pairs (i.e., the service model and the source code) platform models. The platform models are constructed in a hierarchical manner. Software developers can use the service models to create new service-based applications, while the machine (i.e., code generator) use the source code to generate tailored code for specific target device with a specific device capability and configuration. With this, device capabilities and configuration can be managed.

   The method has been partly presented in the paper "PMG-pro: A model-driven method for the development of service-based applications" and is published in the *Proceeding of SDL Conference on Software 2011* [93].