Chapter 1

Describing Service Architectures

This topic has already given rise to numerous works in the telecoms, Web and IT communities. In this chapter, our objective is not to scrutinize these works in exhaustive detail, but to extract the key results and significant points.

What is the point of service architectures? Through the different sections of this chapter, we will see that they allow for a response to the concerns of stakeholders in the design of a service. Early in the process, this is achieved by aiding the decision-makers to identify the major challenges of the service (e.g. technical challenges and functional challenges) and their positioning in relation to each other, including from the perspective of cooperation and cost control. Later in the process, it is achieved by supplying the service's development and deployment teams with a formal and unequivocal statement of requirements concerning the various features to be developed/deployed and the relationships between them. The manager of a development/deployment project can thus reach a clear vision of the tasks to be completed. A service architecture can therefore be considered as the setting for deliberation between the various stakeholders in the design of a service, particularly between the numerous decision-makers (marketing, technical, financial, etc.) and the many development and deployment teams (network, service platforms, etc.). Given that each stakeholder has their own vocabulary, occupation and constraints, this deliberation is publicized by representations (the service architecture) and by people (the architects of the service).

Let us now specify the content of these service architectures, which we will find in different forms later in this chapter. The term "architecture" is usually defined in dictionaries as "the organization of the components that make up a system". "Organization" is defined as the "way in which a whole is constituted for its functioning". We could therefore define architecture as the way in which a system is constituted by basic components for its optimal functioning, taking into account technical and economic constraints. Architecture is the response to requirements (services rendered, cost, reusing existing features, etc.); these requirements being fulfilled due to the identification of the constituent components in connection with each other. In each section of this chapter, we will therefore return to this search for constituent components in connection to fulfill requirements.

In this chapter, we will deliberately not deal with the term "service" in depth. We will point out the significance of the term within each research community, but we will not seek to analyze it in global terms; this will be the objective of Chapter 2.

1.1. The telecommunications community

1.1.1. *The service and global functional planes of the intelligent network*

The world of telecoms has substantial experience of service design. It can take us back to the concept of service in the 1970s with the switches of the public switched telephone network (PSTN). The behavior of these switches is defined by a state machine, defining the basic telephone service. Little by little, telecoms operators and equipment manufacturers made possible modifications to this basic behavior, triggered by various factors. These modifications were called supplementary services. They were first implemented in the switches' code. Then since the end of the 1980s, in order to minimize delays and reduce development costs, the modifications were also implemented in external entities, through what was called the Intelligent Network (IN). The first service architectures that were distinct from telecommunications infra-architectures were introduced in the 1980s with the concept of the IN, and commercial deployments began at the beginning of the 1990s for fixed and mobile networks.

The basic principle of the IN is to ask the switch, under certain circumstances, to interrupt its default processing in a particular point of its state machine in order to call on another service platform, and then to

interpret the commands received from that platform. The IN thus introduced a dissociation between network and services, as illustrated in Figure 1.1, taken from the "standard Q.1201" [ITU 92a].

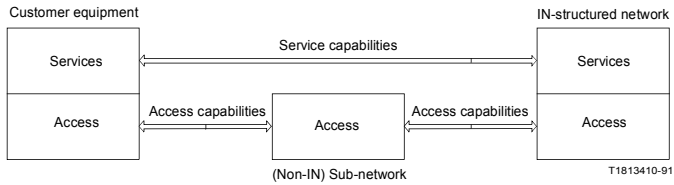

Figure 1.1. *Separation of service and access*

The basic principle of the IN is to separate the call control plane (the switches) from logic and service data held in a service platform. In the IN model, a new functional entity is introduced in the switches, the Service Switching Function (SSF), which has the role of interfacing the switch's resources with service logic held in a service platform called Service Control Function (SCF). The protocol used between the SSF and the service platform belongs to the Intelligent Network Application Part (INAP) family, and allows the service platform to have a precise view of the network and to control it to some extent. Triggering mechanisms allow the service platform to have call control, or to be notified of certain events.

In separating the control plane from the service logic, the IN architecture was the precursor to the Next Generation Network (NGN), which we will discuss later.

To implement this principle in a normalized manner and to facilitate service designs according to this method, the IN community defined a "conceptual model" (called INCM, Intelligent Network Conceptual Model, in ITU-T Q.1201). This model consists of four planes, each plane corresponding to a different structural view (as discussed in section 1.3.1):

– The service plane [ITU 97a] describes an IN service such that it can be seen by a user of the service, for example a freephone number, call forwarding, speed dial or credit card calling.

– The global functional plane [ITU 97b] describes the course of a service, according to a formal method, through a chain of formal components called Service Independent Building Block (SIBs).

– The distributed functional plane [ITU 93a] no longer describes the course the service rendered, but the service's software implementation. A service's implementation architecture defines the entities that have been implemented, linked by protocols and their behavior.

– The physical plane [ITU 93b] describes the implementation of the distributed functional plane on physical machines.

The service and global functional planes allow for the definition of services; the distributed functional and physical planes allow for the definition of their logical and physical fulfillment in a particular environment. Let us discuss in detail the planes that are specifically dedicated to service architecture: the service functional plane and the global functional plane.

In the service plane, there is a composition of service features. For example, a freephone number service could be composed of a "call distribution" service and a "queuing" service. The service plane is presented in the following way in "standard Q.1202", defined as:

> The service plane illustrates the fact that services guaranteed by the IN can be described to the final user or subscriber with the help of a set of generic blocks called 'service features'. (…) A service feature is a specific aspect of a service which can equally be used in conjunction with other services or service features within the framework of a commercial offer. It is either an essential part of a service or an optional section offered in order to improve the service. The service plane represents a view exclusively oriented towards services. This view does not contain any information about the implementation of services in networks (for example, an IN-style implementation cannot be seen). *All that we perceive of it is the behavior of the network linked to the service, in the way that this behavior will appear, for example, to a user of the service*. [ITU 97a] (author's emphasis in italic)

As the last sentence indicates, the architecture of an IN service is incorporated in the service plane from an external view of the service, and not from the internal functioning of networks or platforms. In this first plane, it is the perception of the service by its users that is fundamental. The proposed architecture is summarized in Figure 1.2.

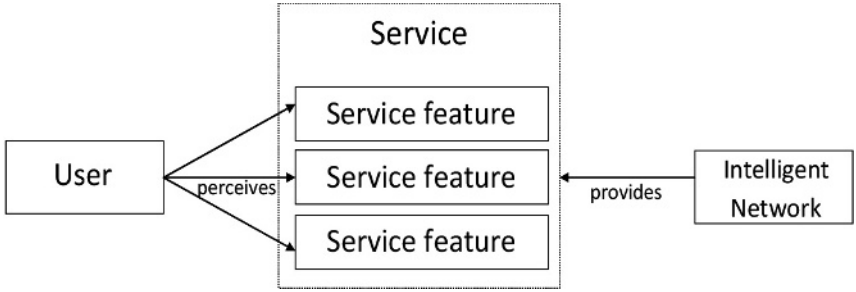

Figure 1.2. *Architecture of service features*

The IN's standardization effort is largely based on network mechanisms, both in fixed networks [Q.1214, Q.1218] and the mobile network [GSM 03.78, GSM 09.78]. In both cases, standards define the behavior of network features such as switches, service platforms and the relationships between these.

In the 1990s, the deployment of second-generation mobile networks gave rise to new demands regarding the IN. One of the fundamental ideas of the Global System for Mobile Communications (GSM) is roaming, which allows a mobile user to use coverage from a network other than his/her usual operator, as is the case with international roaming, for example. The interface between switch and service platform which was until then half-open – that is, open but not specified – therefore had to become fully specified and open. Operators whose network consisted of switches from different manufacturers also expressed the same need to deploy a homogeneous service in a heterogeneous network. The protocols (of the INAP family) and procedures have since been specified in great detail in the standard CAMEL [GSM 03.78], allowing any switch to be controlled by any service platform. This description is primarily based on a Specification and Description Language (SDL, [ITU 92b]) formalism adapted by the European Telecommunications Standards Institute (ETSI). These works have, however, remained at the level of interface between the service and functional planes, defining the interactions between the service platform and networks. In the majority of cases, services or service features are not defined in this case in mobile networks. One reason for explaining this absence is the necessity for the builders of these standards, manufacturers and operators, not to reveal their strategy of service development.

Following on from [MAG 07], we can still consider this model as a precursor to the approaches to service composition that is currently in fashion, for example through the so-called Web 2.0 services.

However, this breakdown of services into service features has never been fully exploited in the IN community. First, the Q.12x2 specifications remain silent on how to identify service features beyond the criterion of common sense (i.e. trying to identify common points from known services). Second, these same standards do not describe how to use service features in lower planes to arrive at a logical architecture, particularly how to transmit service composition in lower planes. Third, manufacturers have not used support from these service features to specify their services from the point of view of users. As N. Simoni highlights in [SIM 07]:

> The standard cites [service features] almost as an example, without defining the rules of architecture, nor those of service composition.

Work on the service plane has ultimately concentrated to a large extent on the problem of service interactions, as brought to light, for example, in [KEC 98] – a service interaction designating an undesired behavior occurring when several services are triggered during the same communication. Starting from the definition of teleservices (such as the basic call), and of supplementary services that are compatible with teleservices but whose compatibility is to be determined case by case, formal mechanisms of service description have been developed, for example in SDL. These service descriptions have been strongly oriented toward the detection, in principle, of the incompatibility between services, but have had mixed results as detailed in [GOU 06]. These descriptions have therefore weighed little on the service plane, where only highly visible cases of incompatibility can be detected, but have concentrated on the other planes.

Let us move on to the global functional plane. On this plane, a service is described as a chain of components, the SIBs. This plane is defined by the "standard Q.1203" [ITU 97b]:

> The global functional plane models the functionality of the network from a global point of view. (…) In this plane, services and service features are redefined in terms of large network functions that are necessary for their support. These functions

> are not specific to services or service features (SF) and are called service independent building blocks (SIB).

SIBs are not a refinement of service features. They are independent of services and they model, as indicated in the first sentence of the above definition, the "large functions" rendered by the network (i.e. the switches, IN platforms and associated resources).

A SIB is therefore a function of the technical system that supports services, a function that we seek to isolate as unitary and combinable with other functions to meet service needs (described in the service view). A total of 13 SIBs are defined in "standard CS1" [ITU 93c], including, for example, a charge for determining the charging for a call, the screen for comparing a value with others on a list or queue for call queuing. These 13 SIBs would have to suffice for describing all possible services from the service features of the service plane. However, with SIBs being defined as the abstraction of a network's functionalities, the specificities of different manufacturers have given rise to an excessive number of proprietary SIBs on top of standardized SIBs. Furthermore, the standardized SIBs were only approved in standardization after the first IN implementations, while proprietary SIBs were already in use, thus considerably reducing their impact. Finally, given the significant fixed costs related to the IN, there was probably a limited number of profitable services that could be implemented.

One of the big questions left open by the conceptual model of the IN is, without doubt: How to move from the service plane to the global functional plane, that is how to translate a description of the behavior of a service as perceived by the user into a description of the behavior of the network? In the case of the IN, this translation was left to the service developer's expertise. Through the global functional, distributed functional and physical planes, the intended move was from the specification of a service to the software code. In fact, several examples of both industrial and academic works, for instance as described in [NAJ 99], have focused on a service creation environment allowing for the transcription of a specific service in the form of an SIB sequence code that can be used on IN platforms. Such a perspective has, of course, led to a more complex global functional view, as SIBs must not only describe a service's architecture but also accurately specify its behavior in order to derive an implementation. Ideally, service development work would also have been taken to the global functional plane level. The value added by the service developer was then precisely within

the transition from service plane to global functional plane, and in theory the other transitions could be automated. In practice, this approach has never been made operational. On the one hand, as we have said, the aim of automation requires the service description to become more complex, making it a *de facto* service code and also the IN expert's prerogative; on the other hand, this automation has never been fully achieved since manual recovery/transfer of the final code is always necessary.

In this work, we will retain the distinction between service description in the service plane and network behavior modeling in the global functional and distributed functional planes of the IN. We will not go into detail here about service creation, for example by the generation of a code from a service architecture description, but will limit ourselves to the consideration of several viewpoints to describe service architectures, which will constitute a reference to development teams.

1.1.2. *From TINA to the NGN*

The Telecommunications Information Networking Architecture Consortium (TINA-C) initiative, described, for example, in [INO 98], attempted to surpass the IN's limitations but did not address the issue of the transition from service plane to global functional plane. The concept of a generic service session, independent of the executed services[1], has instead been defined – a session being a temporary relationship between objects for the completion of a task within a given time (these objects will be both an abstraction of features perceived by users and of technical features). The first requirement of TINA-C is as follows:

> 1: Support of a wide range of services. The TINA service architecture has to support telecommunication, management and information services and should be open to allow the introduction of new classes of services. The service architecture addresses the evolution of services, and should be able to support new requirements and business needs. [INO 98]

To fulfill this requirement, several roles and several session types have been defined, as illustrated in Figure 1.3.

1 In doing so, it may have announced the NGN, which will be specified without an explicit vision of service as we will see in Chapter 2.

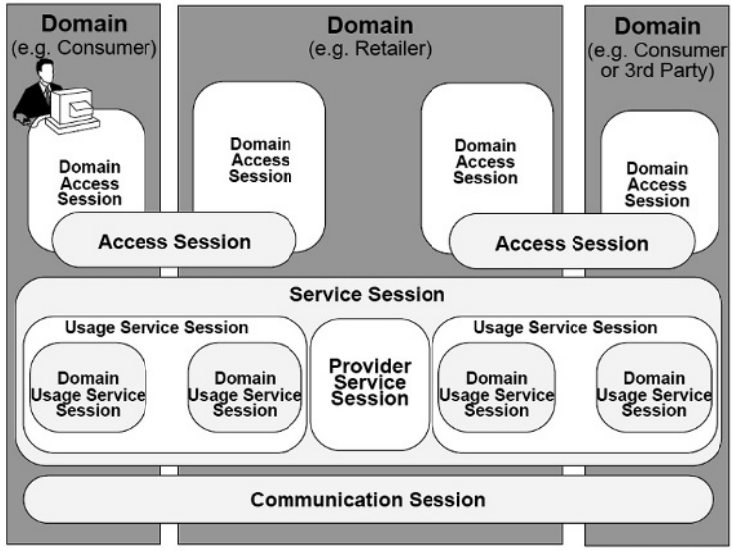

Figure 1.3. *TINA-C session architecture*

These works are doubtlessly closer to the definition of a service marketplace than of a service architecture. It is also surprising to note, following [MAG 07], that these concepts did not lead to the deployment of applications in the telecoms world, but in fact found solid application in the IT world with service-oriented architectures (SOA). TINA-C also opened the path for the consideration of the IT world's methods in telecoms services, particularly object methods and software components.

On the International Telecommunication Union – Telecommunication Standardization Sector (ITU-T) side, work has been carried out on the Global Information Infra-architecture (GII) in order to bring the telecommunications, information and entertainment industries closer together, as presented, for example, in [ASA 96] or [MOR 98]. As far as services are concerned, the Y.110 standard [ITU 98a] defines a typology in layers of service, classified according to the functions that support them. In this standard, we can distinguish:

– baseware functions, including information transportation functions and processing and storage functions;

– middleware functions, based on baseware functions, including application/service creation functions.

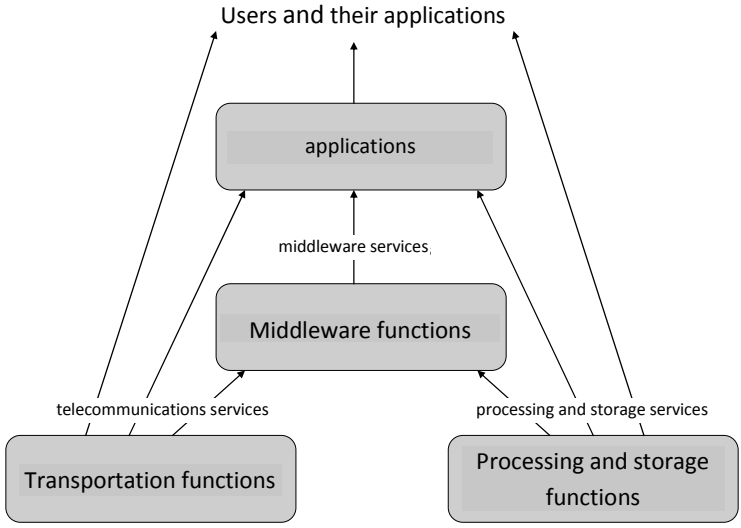

Figure 1.4. *Typology of GII functions*

In Figure 1.4, the arrow from function A to function B means that function A supplies a service to function B, and therefore that function B uses function A. Information transportation functions are, for example, those of an Integrated Services Digital Network (ISDN), PSTN or the Internet. Processing and storage functions can include those of a PC, STB (set-top box), file server or video server. Middleware functions include security, format conversion, authentication, billing and conference bridges. However, this model does not come with instructions on the conditions of its validity and lifespan, and progress in telecoms services in recent years would require its modification. It is probably lacking in uniformity between baseware (abstraction of transport resources and processing/storage of information) and middleware (which are transverse functionalities between services and resources) functions. This division could therefore be used to describe a service, but it cannot really describe a service architecture.

The ETSI has also distinguished different service types. In the technical specification TS 22.101, the last version being [3GPP 09], communication services are divided into conversational services, messaging services,

retrieval services and broadcasting services, whether this is controlled by the user (e.g. with video on demand) or not (e.g. with television). However, this distinction remains high level and there is a lack of explanation of its origin and conditions of validity.

More recently, as part of the NGN, which will be described at the beginning of Chapter 2, several initiatives have grown around services. There have been debates [COC 02] to determine whether the NGN should standardize services, or whether the services should remain exclusively the responsibility of service providers who use the NGN. The economic model of European operators as service providers (not just providers of network infrastructures) and their wish to distinguish between themselves through services, linked with the desire to "cash in" the NGN deployments with services, has led several standardizing bodies to take a greater interest in services.

1.1.3. *The OMA and the concept of the enabler*

The Open Mobile Alliance (OMA)[2] is a consortium, founded in 2002, that brings together telecoms and IT industries with the following aim:

> The mission of the Open Mobile Alliance is to facilitate global user adoption of mobile data services by specifying market driven mobile service enablers that ensure service interoperability across devices, geographies, service providers, operators, and networks while allowing businesses to compete through innovation and differentiation.

This objective is a good summary of the position of standardization organizations in relation to services. To avoid standardizing services and facilitate the possibility of a differentiation between operators in terms of something other than tariffs, the OMA, and also the ETSI and ITU-T, opted to standardize service capabilities, called service capabilities by the 3rd Generation Partnership Project (3GPP), service support capabilities by the ITU-T and service enablers by the OMA. The ITU-T's service support capabilities typically include [CAR 05] presence, localization, group management, message management, broadcast/multicast or equipment management. The ETSI's service capabilities are, for example, presence

2 http://www.openmobilealliance.org/

[3GP 07a], messaging [3GP 07b] or conference calling [3GP 07c]. The OMA's service enablers bring together [OMA 07b] data synchronization, equipment management, electronic rights management, downloading, email notification, instant messaging, presence or mobile localization. These features, which for the sake of simplicity we will collectively call enablers, are reusable application modules, whose behavior toward their environment is fully specified. Theoretically, an enabler from one manufacturer could even be used by another manufacturer's service platform. This approach allows for greater interoperability between equipment, operators and service providers. It also creates the possibility of an improved user experience. Effectively, each enabler has a clearly defined responsibility, for example for data such as user presence or preferences, which means that the user only has to enter this information once for all the services that need it. Indeed, if each service requiring user presence as information were to implement a function to capture and publish it, the user would have to publish his/her presence (or install client presence software) as many times as the services. With a presence enabler, the enabler centralizes this information and distributes it to all the relevant services, simplifying interaction with the user and reducing the network load generated by the use of presence. In more technical cases, such as the synchronization of data between mobile terminal and service platform, the service will not reuse an information but will reuse a synchronization mechanism between the service and the terminal.

Enablers can be seen as building blocks for the construction of services, as presented in the article [BER 04]. While SIBs must be considered as logical operators (e.g. SIB Compare, SIB Screen and SIB Translate), entirely independent of services specified by their sequence[3], enablers are not independent of the offered service but constitute a feature of it, which is reusable in other services. An enabler can be approximated in this sense by a service feature of the IN's service plane. The difference with a service feature lies in the fact that an enabler prescribes not only the service rendered but also its technical fulfillment and the application interface to access it, as shown in the OMA's definitions:

> (an enabler is) A technology intended for use in the development, deployment or operation of a service; defined in a specification, or group of specifications, published as a package by OMA. [OMA 07a]

3 This aspect is also highlighted by the fact that they are called "service independent".

> An enabler should specify one or more public interfaces. [OMA 07b]

An enabler can be seen as a variation in the telecoms world principal of the reusable software component of SOA, which we will introduce in the following section. Work on enablers continues in the Application Programming Interface (API) specification movement for telecommunication networks, notably on the initiative of the Parlay group, founded in 1998. Parlay's initial aim was to specify application interfaces over telecoms call control equipment to allow third-party service providers to fulfill simple services, for example click-to-dial. The group then aligned itself more with the ETSI's Open Service Access (OSA) initiative. Common specifications have therefore been published, most notably for call control, messaging and presence APIs [MOE 03].

An enabler differs nevertheless from an API as from an SOA service, as it is specified not only by its interface but also by its place in an environment, constituted by other enablers, network resources and services, all linked by reference points. Unlike SOAs, it does not have a unique access protocol such as http, but integrates into a complex protocol environment. Furthermore, while the semantics of operations to be fulfilled by an SOA service is unique to that service, this semantics will be induced by the protocol. The behavior of an enabler is also specified in more detail than that of an API. Finally, an enabler is based on precise technology (such as syncML for data synchronization), whereas an API is generally regarded as independent from the underlying technology, but with this often being reflected through the API's input parameters.

In addition, as indicated in the OMA's definition above, an enabler presents a normative character (defined in a specification). Ultimately, a component or a technology constitutes an enabler through standardization. The ITU-T, the ETSI and the OMA therefore do not propose criteria for determining the coverage of an enabler, but a method "standardization", and a concerted standardization between enablers. In effect, if they were specified independently from one another, there would be a risk of the enablers partially overlapping, i.e. that they would offer redundant functions. Service providers would therefore have difficulty combining different enablers to make a service. As shown by the OMA, we would be in the following situation:

> Integration and deployment of services is complicated and expensive; high implementation efforts for applications wanting to use several capabilities; there is no common integration of the different services from the point of view of the end-user (e.g. no common group management or user profile across multiple services). [OMA 07b]

To avoid this, the OMA has introduced the notion of the intrinsic function, which is defined as:

> those functions that are essential in fulfilling the intended task of the specified enabler. For example, the Position Calculation function is intrinsic to Secure User Plane Location; Authentication is intrinsic to Single Sign On; Encryption is an intrinsic function of Digital Rights Management. [OMA 07b]

The OMA specifies that an enabler should only contain intrinsic functions:

> any requirements or features that are not intrinsic to an enabler should not be specified within the enabler's specification.

This requirement gives assurance that different enablers will not offer the same function. This guarantees that, for example, there will not be an authentication function in every enabler, although most do require one. This idea of intrinsic function does not, however, fully deal with the question of the functional coverage of enablers. First, an intrinsic function can mean either a service functionality, perceived by the user – such as a contact group management or user profile function – or the abstraction of a technology – such as a push or watermarking function. Furthermore, the separation of intrinsic functions is not clear. It remains ultimately subjective, as recognized by the OMA:

> The classification of intrinsic and non-intrinsic is subjective and needs to be done on a per enabler basis.

OMA enablers, ETSI service capabilities and ITU-T service support capabilities are important tools for the convergence of services that we will discuss in Chapter 2. They effectively allow different telecoms services to

share and reuse data and functions. But their construction method is heavily linked to standardization. It is due to the contradicting arguments on standardization that their functional coverage (their intrinsic functions) can be determined and legitimized.

How can this method for building converging services be extended within service providers or operators? Can an operator structure his/her services by building his/her own enablers? How? One solution is to implement instances of "standardization" inside service providers, with the same role as in organizations such as the OMA in constructing non-redundant and legitimate enablers. But due to the hierarchical nature of businesses, the debate will probably be less controversial than in the case of standardization that has been developed taking into account different viewpoints and interests. Another solution, which could support the first solution, is to implement rules and procedures – conforming to the functioning methods of big business – on the one hand to identify the service functions that could be the subject of an enabler, and on the other hand to help service development teams to identify the relevant enablers for their projects, and allow them, in principle, to validate the consistency of a service created in this way. In Chapters 3 and 4, we will present an approach that can provide the basis for such rules and procedures.

To conclude this exploration of the notion of service in the telecoms world, let us emphasize two points. First, we need to consider the service plane of the IN and the related notion of service features. These service features are a first step toward reusable service components for the fulfillment of different services. Second, the notion of the enabler, which has extended the service component concept by coupling the service rendered with an effectively reusable implementation. This pairing between service rendered and implementation makes service architectures uniform by imposing just one technical solution to respond to service needs. This in fact leads to a complete standardization of enablers, which restricts their lifecycles (an enabler will take time to be specified and to be implemented). It therefore also restricts the services' lifecycles, if they are supposed to be based as much as possible on the enablers.

We will now examine the description of service architectures in the Web community.

1.2. The Web community

1.2.1. *Web services as fundamental structural units*

From an architectural point of view, the Web can be seen as a group of resources that is accessible via hypertext links. These resources are the fundamental architectural units of the Web. The term "resource" is to be understood in the following way, according to the RFC 3986 *Uniform Resource Identifier (URI): Generic Syntax*:

> The term 'resource' is used in a general sense for whatever might be identified by a URI. Familiar examples include an electronic document, an image, a source of information with a consistent purpose (e.g. 'today's weather report for Los Angeles'), a service (e.g. an HTTP-to-SMS gateway), and a collection of other resources. A resource is not necessarily accessible via the Internet; e.g. human beings, corporations, and bound books in a library can also be resources. Likewise, abstract concepts can be resources, such as the operators and operands of a mathematical equation, the types of a relationship (e.g. 'parent' or 'employee'), or numeric values (e.g. zero, one, and infinity).

In the Web community, a service is thus seen as a resource among others, in the same way as an electronic document or an image, the service being, like any resource, accessible by a URI. Once the fundamental mechanisms of the Web were in place, the Web community started becoming more specifically interested in services in the early 2000s, by specifying mechanisms dedicated to services. The World Wide Web Consortium (W3C) defines a Web service in the following way:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL, Web Services Description Language). [http://www.w3.org/TR/ws-gloss/]

More precisely, a Web service is an application or a software component with the following properties:

– It is identified by a URI.

– Its interfaces and links (binding) can be defined in Extensible Markup Language (XML), particularly in WSDL.

– Its definition (interfaces and service links) are discoverable by other Web services, for example by a Universal Description Discovery and Integration (UDDI) registry.

– It can interact directly with other Web services via XML and the use of Internet Protocol.

A technical reference architecture for Web services has emerged rapidly [CHA 03]. This architecture is made up by the Web service's requester, provider and registry. The Web service provider publishes the services he/she is offering in a Web service registry. The Web service requester consults this registry to find the right service, and then makes contact with the corresponding provider.

As shown, for example, in [KAR 07], these concepts can work in conjunction with those of TINA-C's broker and retailer or with Parlay's framework [MOE 03].

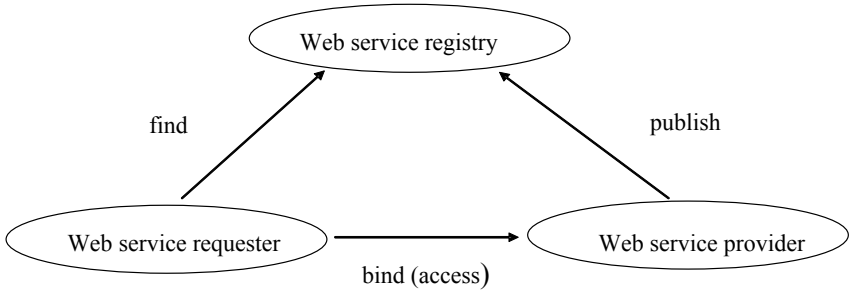

Figure 1.5. *Reference architecture in Web services*

Web services are described by their interface, usually in WSDL, which specifies the service's features, i.e. its input and output parameters. In the Web community, therefore, a service architecture consists of a group of Web services interacting in the pattern shown in Figure 1.5. A broader reflection on the links between services has been conducted, particularly in the context of the semantic Web community.

1.2.2. *Semantic description of resources*

The term Semantic Web, attributed to Tim Berners-Lee [BER 01], covers a vision rather than precise tools. This vision can be summarized in the following aim: enable machines to consider Web-based information sources and take higher level information from them in order to respond to requests. For example, automatically creating a list of French neo-classical painters, or of the works of a particular author, starting out from the raw information available on the Internet. Methods based solely on keyword searching or textual proximity to keywords are not sufficient for these tasks. They do not allow for a distinction to be made between, for example, books by Albert Camus and books about Albert Camus, or between a painter working within a trend or rejecting that trend, as textual proximity to the search terms could come up in both cases. This kind of data manipulation by machines creates a need for an information architecture that can be accessed via the Internet in a formalism allowing for the automatic processing of numerous and varied sources. For this purpose, the semantic Web community has adopted the concept of metadata, a metadatum being a piece of data used to describe another data [BER 01]. All informational resources can be described by metadata. Unlike basic data, metadata follow a semantics that is comprehensible to machines. These metadata are not keywords, but are rather structures. Thus, a painter and his/her alignment with an artistic trend could be indicated by metadata attached to a painting. In order to be efficient metadata will, of course, have to use keywords such as "neo-classical". The Resource Description Framework (RDF) model, published in 1999, allows for the description of resources with the help of metadata, as well as the links between metadata[4]. Artificial intelligence software called inference engines can then conduct logical reasoning using metadata.

An agreement on a selective and coherent list of the metadata to use to describe resources is, however, unrealistic in an environment as broad and open as the Web. The Semantic Web community has therefore also worked on models of knowledge representation. This work has produced ontology languages like the Ontology Web Language (OWL); an ontology being understood, according to T.R. Gruber's [GRU 93] famous definition, as:

4 A metadatum or a link can also be considered as a resource and described with the aid of other metadata.

> The specification of a conceptualization of a field of knowledge.

An ontology therefore defines a set of metadata for the description of a domain, and the relations between these metadata through preconditions. An ontology can therefore take into account the relationships between metadata. They are usually intended to be created and shared by several people working in the same knowledge field. Someone wishing to describe a resource can therefore attach the metadata he/she uses for this to an existing ontology, by indicating the relationships between the metadata and the ontology concepts to which they refer. An ontology can be constructed, and also automatically generated, from a set of representative structured documents, but this can give mixed results as discussed, for example, in [BED 08].

1.2.3. *Semantic description of Web services*

The Semantic Web community has sought to apply these principles to Web services. It has also taken an interest in service description, launching initiatives for the addition of a semantic description to Web services, as shown, for example, in [ZEN 01, ZHI 07] or [ARR 06]. Web services are thus considered as resources and described by metadata, which we refer to as service ontology. Two complementary directions were investigated. On the one side, specifications were proposed for the addition of semantic annotations to WSDL files, such as Semantic Annotations for WSDL (SAWSDL) [MEN 07]. On the other side, other specifications aimed, in a more ambitious but less incremental way, at describing a service in all its different dimensions rather than just its input and output parameters. Two Web services with the same signature could effectively perform completely different services. A service with an input parameter of two telephone numbers could, for instance, just as well be a click-to-dial service as a call forwarding service from the first number to the second. Specifications such as OWL for Services (OWL-S) aim at supporting the construction of Web service description ontologies. A Web service has three dimensions: the service profile, the process model and the grounding. The service profile described what service is rendered, the process model described how the service is used, and the grounding describes how to interact on a practical level with the service. The profile and the process model are abstract characterizations of the service, while the grounding forms the relationship

between these abstract descriptions and the concrete messages exchanged by the producer and the consumer of the Web service, via a given protocol.

Other works have concentrated on the construction of Web service marketplaces. These works aim to create an environment of discovery and access to services, which are described semantically according to coherent ontologies. This would enable the discovery of services from a user's statement of requirement, thanks to the inference mechanisms. In addition, semantic annotations would facilitate the automatic creation of a service to respond to a certain requirement. Works such as [SHI 08] or [LEC 09] propose a correspondence algorithm between the input and output parameters of a Web service, these being described semantically in order to detect the relationships between them (equivalence, affiliation, etc.). A chain of Web services can thus be automatically conceived in response to a statement of requirement in the form of input and output parameters. The semantic Web community proposes an automatization of services' discovery and of service composition tasks, by categorically modelizing the knowledge of humans who could do these tasks. In a target view, users would be able to express (in natural language, for example) their requirement so that a service can be automatically created to respond to it, the marketplace relying for this on the services it registered. This marketplace would link numerous services without formerly agreed consistency, the selection of services and their consistent functioning being assured by semantic technologies (such as semantic annotations, ontologies and inference engines). The notions describing each service and the users (preferences, profile and context) would be defined as ontologies. Reasoning on these ontologies would ensure coherence between these notions. These approaches aim to resolve the difficulties (that we will discuss in the following section) for SOA services, such as service identification, service granularity and service lifecycle management.

In terms of telecoms architectures, work has been undertaken on ontology construction for communication services, such as in the IST SPICE project (http://ontology.ist-spice.org/index.html). Service marketplaces are also the subject of study for telecoms services, as in the CELTIC Servery project (http://projects.celtic-initiative.org/servery/). However, these remain prospective, as they face several obstacles.

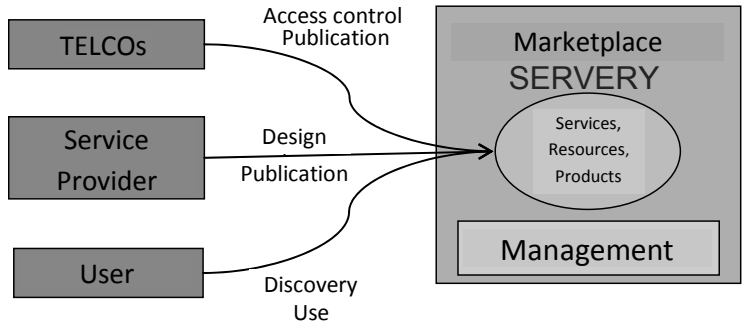

Figure 1.6. *Marketplace concept in the CELTIC Servery project*

First, a problem of critical mass and industrial maturity, as indicated in [HAM 07]. Semantic tools are still in the academic development phase, and ontological construction and inference tools remain largely unused outside of research projects, as their relevance, reliability and ease of use are not currently fully adequate. There is also a scaling problem, particularly in terms of performance and complexity management.

A more fundamental problem is the lifecycle of an ontology. In the semantic paradigm, the relevance of the response to a given need relies on the quality of the ontology that enables the linking of metadata describing services and user needs. How can these ontologies be created and developed? It could be done centrally, by a team responsible for the consistent construction and maintenance of an ontology. But this team might then neglect the problems of completeness, lifecycle, service identification and service granularity that these mechanisms were supposed to resolve. The ontological tool therefore holds only a weak advantage over less developed processes such as the specification of a list of metadata. It could also be done automatically, by detecting the most commonly used keywords in service interfaces and linking these together. But as can be seen in [BED 08], such automatic mechanisms are only relevant if based on a consistent classification, and otherwise will produce inconsistent ontologies. Semantic service annotation also poses a problem. If this annotation is created centrally, by a team responsible for the semantic annotation of new services, the benefit is minimal compared with a more traditional system such as the one advocated for the lifecycle management of SOA services. And in the case of a marketplace providing many services, this team would soon be overstretched. But requiring service designers to mark up their services

themselves by referring to an existing ontology would be extremely restricting. They would effectively have to be familiar not only with semantic tools and languages but also existing ontologies, and be able to form a link between the concepts they handle and those of the ontology, which practically equates to the task of service specification in the semantic form. We often see, as shown, for example, in [HEP 07] or [ROC 07], a lag between the technical vocabulary of service developers and that of service users. Correspondence between the two is not immediately accessible, as it is not about translation or finding synonyms but two different perspectives of the service, without an unequivocal link between the concepts handled by each.

Furthermore, there is a contradiction that can be revealed between the approach of SOAs and the semantic approach. In effect, the semantic Web, as shown, for instance, in [PAT 07], follows a white box or open-world paradigm, i.e. the universe of discourse tends to encompass all services offered, including the detail of their properties [ROL 00]. On the other hand, SOA service platforms, as we will see in the following section, have a black box or closed-world approach, exposing as little as possible of their internal functioning, in order to guarantee a loose coupling between different services.

Finally, the semantic paradigm requires data to be comprehensible to machines, without doubt to the disadvantage of humans. Like some of the formal SDL descriptions in the fixed IN, an ontology is not instantly readable; semantic data are designed first and foremost to be read by machines. This makes it difficult to generalize these methods for service architecture design.

What should we retain from this journey? Unlike the telecoms community, which sought to identify service components independently of each other and then link them to perform a service, the Web community and the Semantic Web community emphasize the importance of links as essential vehicles of knowledge. Services, and even service features or the so-called unitary services, are not completely independent blocks. There are links, prerequisites and dependencies between services and with infra-architectures, which semantic methods seek to accurately capture in ontologies. In this work, we will not position ourselves explicitly in the semantic paradigm, but we will try to respond to the above questions, left

open by this paradigm, via a classification of telecoms services and the links between different perspectives of a service.

1.3. The IT community

The term information technology (IT) refers to the whole of enterprise computing. The Information Technology Association of America (ITA, http://www.itaa.org/) defines the term IT as:

> The study, design, development, implementation, support or management of computer-based information systems, particularly software applications and computer hardware.

1.3.1. *Service-oriented architectures*

In the IT world, work on services is guided by the needs of information systems (IS). An IS can be defined as follows [OBR 97]:

> An information system is a set of people, procedures and resources that collects, transforms and disseminates information in an organization.

Information systems are a key tool in the strategy of a business; their job is to implement business processes by serving as a support to information management. The computerized realization of an IS is called an informatics system. The alignment of the IS with a company's business is a major challenge, as detailed, for example, in [HEN 93] or [SIM 07].

However, the business of services companies is evolving toward a decompartmentalization of organizational entities in order to respond to client needs. This evolution is especially observable in the economic sector of services, as we will see in Chapter 2. IS should therefore evolve from a very compartmentalized "vertical" organization, where each organizational entity manages its IS autonomously as they do not need to communicate or exchange (the business processes they support being unique to one organizational entity), to a more "horizontal" organization, where the IS of the different organizational entities continually make exchanges to support transverse processes between the company's various functions, in order to respond to client needs. Let us consider, for example, the domain of banking

insurance. If, 15 years ago, a banking network could make a client change their account number when they changed branch, today that same client could not only keep their account number but also use it to access various financial and insurance products via multiple distribution channels (branch, call center and Web). This decompartmentalization is also the rule in the e-commerce sector, as shown, for example, in [MOR 03].

The IT world created the SOA concept to facilitate this decompartmentalization. In the SOA paradigm, IS applications are broken down into software services, i.e. independent application units that fulfill a fixed and limited business function, accessible via an open and defined interface; this concept of software service was essentially implemented with the aid of Web services, seen previously. SOA has been defined by its initial proponents, IBM, in [CHA 03] as:

> an application architecture within which all functions are defined as independent services with well defined invokable interfaces which can be called in defined sequences to form business processes.

In the methods of analysis of business processes, such as [SIM 04], following [BOO 99], a business process is understood as:

> a sequence of management actions carried out by a company, which produces a result whose value is perceptible and measurable value to an individual subject of the modeled domain [SIM 04].

In the SOA paradigm, this sequence of management actions can be carried out by a chain of calls to software services, at least when these management actions are computerized. Business processes can also be specified in appropriate language, such as the Business Process Modeling Language (BPML), and this description can be translated into a script of the software services to be linked, expressed in Business Process Execution Language (BPEL), for example.

As well as this concept of software services, the SOA approach also specifies architecture principles. Software services must be:

– highly cohesive, i.e. all the functions offered by one software service must be consistent and not incongruous;

– loosely coupled, i.e. a software service must only handle data from outside its area of responsibility via other software services, and not do so directly.

Different software services can then be put together, i.e. used one after another to create a more complex service. For example, a software service for automatic translation followed by one for sending text messages would give the possibility of sending a translation via a text message. Linking service A and service B is possible as long as service A has an output parameter that is of the same type as an input parameter of service B. Languages such as BPEL allow for the specification of such links, including logical conditions. These scripts or software services are a means of automating business processes.

But in attempting to achieve business processes with the aid of software services, the IT community found that the principle challenge was not technical. Technically, invocation protocols or specification languages for the sequencing of software services are operational. But the main challenge resides in the identification of software services to be built. How can the boundary of these independent application units be defined? Should a software service be fine-grained, with a service identifiable by function, or more coarse-grained, with a set of consistent functions performing a business objective? With fine-grained services there is a risk of overabundance; it could be impossible to manage the lifecycles of every service, and difficult for a designer to determine what services to use. With coarse-grained services there is a risk of a lag between the business objectives that historically governed the establishment of a service and the current objectives. These objectives vary depending on the business strategy and the external environment (regulation, legislation, competition, market, etc.)[5]. And how can independence between services be maximized when the services are coarse-grained? These issues are, in fact, quite similar to those of enablers that we encountered earlier.

5 A section of the IT community has studied the alignment between the strategy of a company, its business processes and its information system, for example [HEN 93, BLE 06] or [SIM 07].

Transposition of the SOA paradigm to communication services is efficient to a certain extent. It makes it possible to provide reusable access to some functions of platforms and telecommunications networks; these functions often being those identified by Parlay, such as click-to-call, geolocation or the sending of text messages. This transposition does, however, face a major pitfall. Software services in the IT world are initially designed to access IS functions, classically data processing functions. Invoking a service means either requesting data or requesting the processing of data. The value of a communication service in itself lies not in its capacity to process data but in its capacity to put things in relation to each other. The heart of a communication service cannot therefore be reduced to a series of requests/responses between machines, as different stakeholders will be implicated in the service, for instance the maker and the receiver of a call in a telephone service. Protocols, such as Session Initiation Protocol (SIP, [IET 02]) or Simple Mail Transfer Protocol (SMTP), which structure exchanges between parties are essential for communication services, but are not included in the SOA paradigm, where protocols are not seen as a support for the invocation of software services. Furthermore, the SOA paradigm does not consider essential aspects of communication services, such as the quality of the human–machine interface, or the network effect (or network externality) based on the use of a given protocol, carrying a given semantics.

In this work, we will propose a method of describing communication services that responds to the concerns raised. The identification of the scope of services from a non-software point of view related to the business of the service provider, then the construction of an application architecture that can take into account both the software services and the usual protocols of the telecoms world. We will now see how the IT community has formalized this notion of a point of view to describe the architecture of a system.

1.3.2. *The concept of view*

Faced with the complexity of informatics systems, the software engineering community has gradually developed concepts to master this complexity. In the 1990s, notably under the impetus of the Object Management Group (OMG), it adopted the concept of view. The “standard IEEE 1471” [IEE 00], for example, defines the concepts of view and point of view in the following way:

> A view is a representation of a whole system from the perspective of a related set of concerns.

A point of view is the specification of the mode of constructing a view:

> A viewpoint is a specification of the conventions for constructing and using a view, a pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis. [IEE 00]

The standard also introduces the attributes of a view:

> Each viewpoint shall be specified by a viewpoint name, the stakeholders to be addressed by the viewpoint, the concerns to be addressed by the viewpoint, the language, modeling techniques, or analytical methods to be used in constructing a view based upon the viewpoint. [IEE 00]

Each view of a system must therefore respond to the concerns of the stakeholders in a system. For an IS, these stakeholders will be, for example, the final user of the system, the architect, the developer or the manager.

This concept of point of view has been employed in numerous frameworks. One of the most famous is without doubt the 4 + 1 framework proposed by Kruchten [KRU 95]. As the name suggests, it proposes four points of view on the system, linked by scenarios illustrating the system's essential functionalities.

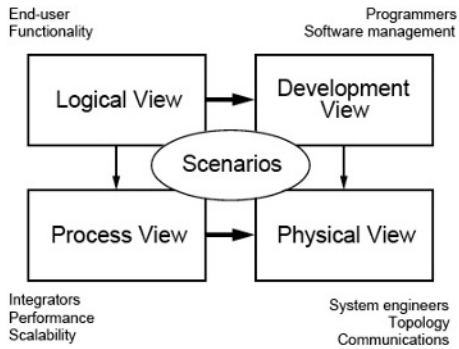

Figure 1.7. *Points of view in the 4 + 1 framework [KRU 95]*

The attributes of these views (as presented) are described in Table 1.1.

Point of view	Logic	Process	Development	Physical	Scenarios
Objective	Indicate the service that the system is to provide	Indicate the competition and synchronization aspects of the design	Describe the static organization of the software	Describe the implementation of software onto hardware	Discover the key concepts of the design and validation
Stakeholders	Architect, user	Architect, designer, integrator	Architect, developer, manager	Architect, designer	Architect, user, developer
Concerns	Functionalities	Performance, availability, etc.	Organization, reusability, portability, etc.	Scalability, performance	Comprehension
Modelization concepts	Classes, associations, heritage	Events, messages	Subsystems, components	Machinery, bandwidth	Classes, events, stages

Table 1.1. *Points of view in the 4 + 1 framework*

Following this work, the concept of the point of view will play a key role. It is comparable with the IN's plane concept, but clarifies it. While the IN planes can be seen as a layered architecture, with designers only considering the plane above theirs, the concept of view is more flexible and specifies the criteria for the constitution of a view, as a tool for responding to the concerns of the stakeholders in a system. We will now examine how this concept of view has been applied to IS.

1.3.3. *Enterprise architecture and urbanization*

Enterprise Architecture (EA) and urbanization methods emerged before SOA methods, as a variation of the concept of view in the framework of business IS. But these methods remain relevant for implementing SOAs, as they aim to take on the major difficulty discussed above, i.e. the implementation of software services that are consistent with the company's strategy.

EA methods aim to facilitate the evolution of business IS. They offer a representation of the IS through different views, in order to make it comprehensible to the decision-makers while also ensuring a certain amount of consistency between the decision-makers' view, the development teams' view and the hosting teams' view. They generally work within four key steps, as shown in [SIM 09] or [URB 06]:

– mapping what exists, i.e. identifying and describing the company's resources (human or software) and their positioning in the IS, as well as their responsibility for the information handled by the IS;

– defining an ideal target that fully conforms with the company's strategy;

– moving toward the target, i.e. setting tasks to achieve within a certain time in order to move the existing IS closer to the target;

– itemizing these tasks in development projects. The architecture that is implemented in these projects must therefore fit in with the target.

In the British and American works, numerous EA methods have been proposed, including frameworks by Zachman [ZAC 87], the Department of Defense Architecture Framework (DoDAF, http://dodcio.defense.gov/Portals/0/Documents/DODAF/DoDAF_Volume_I.pdf) [DoD], The Open Group Architecture Framework (TOGAF, http://www.opengroup.org/togaf/) and the Information Technology Infra-architecture Library (ITIL, http://www.itil-officialsite.com/home/home.asp). There is, thus, such an abundance of EA methods that some [SCH 03] describe this as a jungle. These different methods are generally defined with a liberal perspective. They are relatively unrestrictive (for the modelization of points of view, for example) and do not seek to hierarchize and systemize this modelization, the most important thing being the dynamics that are sparked in the company [SAL 08].

In France, a more constrained framework has been born with the urbanism of IS. Urbanization emerged in big businesses during the 1990s, particularly in the banking sector, faced with controlling their IS. Jacques Sassoon, the first urbanization project manager at *Crédit Agricole* and later the *Société Générale* popularized the application of the city metaphor to IS [SAS 98]. From this perspective, an IS can be structured, like a Haussman-style town, into zones, districts and blocks, linked by communication routes. This division into blocks, as well as the rules of construction applicable to

each block, can be rationally documented in the image of a British Local Development Plan (formerly the Local Plan). A validation committee can then authorize or not authorize each modification project of an IS depending on whether or not it respects the rules that apply to the block in which it is located. These principles were completed, notably by Christophe Longépé, who was in charge of urbanization at Sema and then at the *Société Générale*, by formalizing views, with a model and rules associated with each view. He also distinguishes four viewpoints on the IS in [LON 06], the division into blocks being essentially the prerogative of functional and applicative views:

– "The business process view, that describes the business processes and also the relationships between them;

– the functional view, that describes the functions that the information system has to support;

– the applicative view, that describes all the software elements of the computer system that automates the information system;

– the technical view, that describes the overall technical architecture."

In other words, the business view responds to the question of "why", the functional view responds to the question of "what", the technical view responds to the question of "with what" and the applicative view responds to the question of "how" [SIM 09]. Urbanization is therefore defined as the process of applying these principles to a particular business.

> Urbanization is to organize the gradual and continuous transformation of the information system, to simplify it, to optimize its added value and to make it more responsive and flexible towards strategic business changes, while relying on technical opportunities of the market. It defines rules and a coherent, stable and modular context, in which different stakeholders are referring to any investment decision in the Information System. [Urba-EA]

As highlighted by Contini [CON 02], urbanization rests, unlike EA methods, on the presupposition that sufficiently stable invariants can be identified to hierarchize the blocks and enforce rules that will endure, if not forever then for a long time. In fact, in his first urbanization plan for the *Société Générale*, J. Sassoon considered bank products, basic units offered by a bank, to be sustainable. C. Longépé formalizes this concept by

considering several types of invariants: those coming from the business view, called concept classes; those linked to the aims of a business IS (an exchange zone, a data repository zone, a referential zone for data and rules, a steering zone, an operations zone and a resource zone); those coming from the technical view, such as the three-tier model of technical architecture. This concept of the invariant is not immediately compatible with the agile enterprise model, constantly adapting to external developments to maximize its profits. A problem, but also without doubt an interest, of urbanization lies in the identification of these invariants in the company's business, beyond tactical developments. By applying this process to telecoms services in Chapter 3, we will further discuss this question of invariants for communication services.

The principles of EA and urbanization have already been applied by telecoms operators – not to telecoms services, but to the internal activities of operators or telecoms service providers. In other words, the task of defining a framework and the organization of an IS was carried out on services' fulfillment, billing and supervision, but not on the use of these services. Importantly, to telecoms operators the term "information system" denotes systems that carry out these actions of fulfillment, billing and supervision, but not the service platforms that provide the service. Historical reasons explain this. Operators have long provided only network services, i.e. the establishment of a connection. The key challenge was therefore the implementation of this connectivity at the network equipment level (via their provisioning), billing and supervision[6]. In their urbanization processes, operators therefore focused on their internal processes, and not on the processes that they shared with their clients.

These tasks have been notably executed by a standardization organization called the TeleManagement Forum (TMF). The key areas addressed by the TMF in this sector are the definition of a reference framework of business processes, the definition of a set of standard business processes, the definition of systems for the realization of business processes and the implementation of solutions. The TMF has therefore formalized the enhanced Telecom Operation Map (eTOM) model, summarized in Figure 1.8. For more details, see eTOM (http://www.tmforum.org/BestPracticesStandards/BusinessProcessFramework/1647/Home.html).

6 Recent work has, however, applied the principles of urbanization to network services, notably F. Menai in [MEN 05]

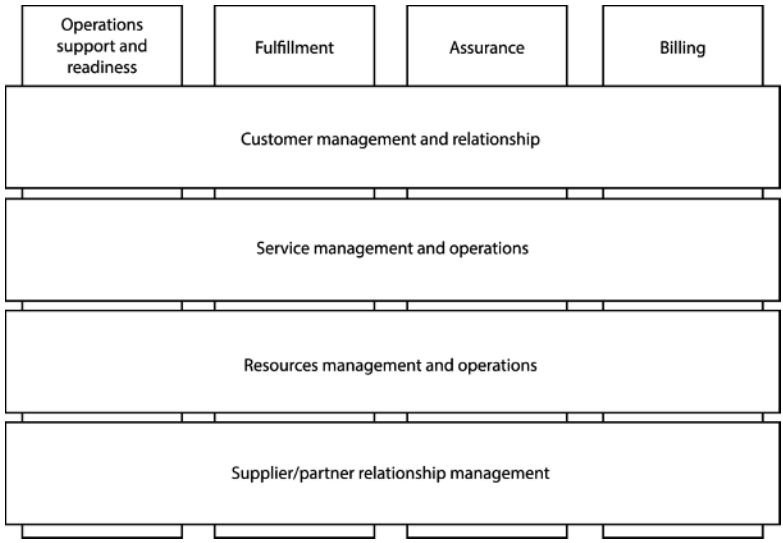

Figure 1.8. *eTom reference framework*

The rows and columns in Figure 1.8 constitute the invariants that we just mentioned for telecoms service management. The rows represent the main domains of the IS, which can be compared to areas of urbanization. These are the customer relationship, service management, resource management and supplier management. The columns represent the main groups of business processes internal to telecoms operators:

– fulfillment, to sell and implement a service;

– assurance, to guarantee the quality of the desired service;

– billing, to invoice the service that has been provided;

– operations support and readiness, to ensure the smooth running of all the above processes.

As we have indicated, this work has not addressed what the value of a service brings to its users, i.e. the use of this service. Service platforms are seen as resources, in the same way as a router or a switch. But service platforms are stakeholders in a system. They support interactions with users. And they interact with other service platforms and with networking equipment, so that a service is guaranteed from end to end. The usual questions concerning the urbanization of IS, such as arranging in blocks,

dividing functional responsibilities between these blocks and the means of communication between themselves, fully apply to current service platforms, of which the architecture needs to be structured and streamlined. We will see in Chapter 2 how this need of the architecture arises from the evolution of communication services to user-centric services, centered on the user. Unlike the approach proposed, for example, in [VUD 03], an extension of eTOM to take service use into account does not seem relevant. The invariants of services are not effectively the same as those of internal service management processes, which are summarized in the columns and rows in Figure 1.8. In Chapter 3, we will determine what is stable in rapidly evolving services.

1.4. Summary

At the beginning of this chapter, we defined architecture as the way in which a system is constituted by basic components for its optimal functioning. What are these basic components in the telecoms, Web and IT communities? They are, without doubt, network functionalities for IN services, service platforms for NGN services, resources for the Web and software services for SOAs.

If these components are defined case by case and without a fixed method, we could soon fall into the traps of over-abundance, confusion and lack of control. With urbanization and EA methods, we have seen that to orient the conception of these features toward a company's strategy, we must identify a target, as well as invariants to use as a basis for identifying these components, to describe their relationships. The search for these invariants is not the prerogative of urbanization, but crosses the telecoms, Web and IT communities. These are the service features and SIBs for IN services, enablers for NGN service platforms, ontology concepts for Web resources and business process activities for SOA services. For communication services, the search for invariants often lacks method, partly due to not showing how to manage their lifecycle (why and how to introduce or remove them?) and partly due to not always clearly distinguishing between the different aspects of a service (functional, technical and applicative). We will now see in Chapter 2 what the basic features that make up telecoms services are.