Characterization of the Dynamic Service Generation Concept

Clement Jonquet, Stefano A. Cerri

Characterization of the Dynamic Service Generation Concept

Clement Jonquet and Stefano A. Cerri

February 2006

LIRMM, CNRS & University Montpellier II
161 Rue Ada, 34392 Montpellier Cedex 5, France
{jonquet,cerri}@lirmm.fr

Abstract

The goal of this position paper is to reflect about the concept of service in Informatics. In particular, we present in this paper the concept of dynamic service generation as a different way to provide services in computer-mediated contexts: services are dynamically constructed and provided (generated) by agents (human or artificial) within a community, by means of a conversation. This process allows services to be more accurate, precise, customized and personalized to satisfy a non predetermined need or wish. The paper presents an overview of the concept of service from philosophy to computer science (service oriented computing). A strict comparison with the current popular approach, called product delivery, is done. The main result emerging by these reflections is a list of "characteristics" of dynamic service generation, in order to promote a progressive transition from product delivery to dynamic service generation systems by transforming one-by-one the outlined characteristics into requirements and specifications. More specifically, two major characteristics are precisely described in the paper as they imply 80% of the other ones. They promote a substitution of an *agent* oriented kernel to the current object oriented kernel of services as well as the Grid as the service oriented architecture and infrastructure for service exchanges between agents.

Keywords: Service, Dynamic Service Generation, Agent, Service Oriented Computing/Architecture, Web service, Grid Service.

Contents

1 Introduction

In [Cla05], Clancey reflects about a personal experience occurred when he had to organize a journey. Using a travel agency service, he did not get satisfaction because his goal was to plan and have a pleasant journey whereas the travel agency people and Web sites wanted simply to sell him tickets. His need was feasible, but none of the agents (human or artificial) that interacted with him has considered his real problem. All of them concentrated on the product that they had to sell: tickets (”piece-meal services”). From this scenario, Clancey explains that ”constraints and preferences usually cannot be formulated up front; the process of articulating constraints is necessarily interactive and iterative” and ”joint construction is required, a conversational interaction that bit-by-bit draws out from me what I am trying to do”. He raises some principles that should be taken into account in (human or artificial) service systems such as for example:

- co-construction of the obtained product, based on a conversation (interaction and iteration);
- real understanding and consideration of the user problem(s);
- trust delegation that must lead to satisfaction;

- pro-activity etc.

As another real life scenario, imagine someone looking for a job in a job agency. To find or to choose a job has so much impacts/consequences on somebody's life that it cannot be generalized, it must be done case by case. Requirements, constraints and conditions are all interleaved one another. The service delivered by the person in the job agency is "dynamically generated": the job searcher would explain during an interactive conversation his/her qualification, diplomas, interests, wants as well as his/her private constraints (family, culture etc.). The service person at the job agency and the job searcher construct little by little a solution that fits for both of them. This solution will have relevant external consequences. This is a kind of dynamically generated service.

The notion of service is now at the centre of development of distributed systems; it plays a key role in their implementations and success. One of these successful systems is of course the Internet, but nowadays surfing on the Web is equivalent to ask another computer on the Internet to execute a calculus specified by an algorithm, or to give an answer to a well specified question. Actually, the Internet allows to offer/deliver to human or artificial entities "ready made informational products", more than real "on the fly generated smart services". We can for the moment store and retrieve "products" (e.g. pages, software, images, function application results etc.) but not yet to participate in an interactive society. More generally, nowadays computers, operating systems, software, provide to their user precise products for which they have been conceived for. A file system enables to create, copy, move, or delete files, a music player permits to listen, record, copy, download music, a railroad or yellow pages web sites allows to buy a train ticket or to find somebody's coordinates. All these "services" correspond to well identified and precise needs or wants and they are specifically constructed to answer them. As the needs or wants evolve, new versions of these systems are updated. It is the classic software engineering life cycle. However, could we really talk of services?

To provide a service means to identify and offer a solution (among many possible ones) to the problem of another. We try in this paper to explain how a service in a computer-mediated context may not be reduced to a simple "method invocation", how it is unique and how it needs a real engagement of entities using and providing this service. We argue that a real service providing is not a product delivering, because a service is adapted, customized and personalized by a special provider for a special user. A very important aspect of the concept of service is that it should be based on a conversation. A service should be seen like a kind of collaboration. The paper introduces the concept of Dynamic Service Generation (DSG) as the process of obtaining services constructed on the fly by the provider according to the conversation it has with the user. These services are called dynamically generated services. In DSG service providers and service users are both agents (human or artificial) members of a community.

Actually, DSG is opposed to the classical Product Delivery (PD) approach. The notion of product refers to the idea of obtaining the delivery of something already existing, or already designed, whereas the notion of service relates to the idea of action or creation of something new. The difference is very easy to understand in human everyday life, for example someone looking for clothes may look for *ready-to-wear clothes* which is analogue to asking for a product, or either may look for having *clothes made by a tailor* which is analogue to asking a service to be generated. As another example of DSG, take for instance the one of Singh and Huhns in their recent book on service oriented computing [SH05]: "Imagine going to a carpenter, a human service provider, to have some bookshelves built in your home. You will typically not instruct the carpenter in how to build the shelves. Instead, you will carry out a dialog with the carpenter in which you will explain your wishes and your expectations; the carpenter might take some measurements, check with some suppliers of hard-to-get materials (maybe some exotic woods), and suggest a few options; based on your preferences and your budgetary constraints, you might choose an option or get a second estimate from another carpenter."

Providing these kinds of services in computer-mediated contexts is a real challenge. In spite of the fact that apparently most current literature concerned with services come from the Web communities (i.e. Web services) we are convinced that the challenge cannot be seriously taken unless profiting from results emerging from several domains in computing. The notion of service has to surpass HTTP protocols, Service Oriented Architectures (SOA) standards, remote procedure calls and Extensible Markup Language (XML) to be enriched by other research domains such as information systems, concurrent systems, knowledge engineering, interaction and especially by distributed artificial intelligence and multi-agent systems (MAS). Simply delivering products is not sufficient anymore, a service provider cannot simply execute a single task for a given kind of user as it the case in the client/server mode or with Web services. DSG does not assume the user knows exactly what the service provider can offer him. He[1] finds out and constructs step by step what he wants as the result of the service provider

[1]DSG users are agents, AA or HA, the terms he/him will be used hereafter generically for she/her and it/its.

reactions. He does not exactly know what he wants and his needs and solutions appear progressively with the dialogue with the service provider until he obtains satisfaction. DSG is an interactive process between a user and an information system that realise the DSG, called a DSG system.

Considering interaction between users and information systems, three presuppositions are generally held to be true:

- The user knows exactly what he wants;
- The user knows that the system can answer him;
- The user knows how to formulate his request.

However, none of these presuppositions are verified in DSG. They all lie on the user knowledge and not on the information system's ability to take the initiative, to understand or to adapt to a specific request. DSG systems should not presuppose skills or abilities of their future users (human or artificial).

The idea of DSG proposed in this paper comes from a reflection on the concept of service that appears within both industrial and academic computer science. New needs in service exchange scenarios are clearly highlighted by prominent computer scientists:

- "A paradigm shift is necessary in our notion of computational problem solving, so it can better model the services provided by today computing technology", Wegner [GW04];
- "Often, when services are described, there is an emphasis on invoking services as if they were no more than methods to be called remotely (...) Likewise, in computing, instead of merely invoking a particular method, you would engage a service (engagement goes beyond mere invocation)", Singh and Huhns [SH05];
- "Computers will find the meaning of semantic data by following hyperlinks to definitions of key terms and rules for reasoning about them logically. The resulting infrastructure will spur the development of automated Web services such as highly functional agents.", Berners-Lee [BLHL01];
- "On-line services that "computerize communication" can be improved by constructing an activity model of what the person is trying to do, not just selling piece-meal services (...) We need to provide much better services than what people currently receive.", Clancey [Cla05].

The ambition of this paper is not to specify and formalise DSG systems i.e. systems able to realise dynamically generated services but rather to identify a non exhaustive list of characteristics DSG systems should have. The proposed methodology for DSG system development consists in a progressive evolution from current PD systems to DSG systems by integrating one-by-one these characteristics. 80% of theses characteristics are implied by two important ones that DSG systems must consider (n°5 and n°9, see section 5.2.3 and 5.2.5):

- The substitution of an agent oriented kernel to the current object oriented kernel of services. The agent paradigm is the only approach in computing able to deal with conversation based services.
- The use of a Grid based service oriented architecture and infrastructure to host agent services exchanges. The Grid is the only approach in computing able to deal with stateful based services.

The rest of the paper is organised as follow: section 2 present the concept of service, what is a service, what is a product. This section makes an overview of the intrinsic and social ideas behind the concept of service from philosophy to computer science (service contract, quality of service, etc.). Section 3 is dedicated to Service Oriented Computing (SOC) and presents a state of the art of the main current framework that implements SOA: Web services. A brief taxonomy of Web services is proposed. Section 4 present precisely the idea of Dynamic Service Generation (DSG) by defining the related concepts and presenting examples. A definition of DSG is done by opposition to the main current approach in service exchange scenario, called Product Delivery (PD). The core of the paper is section 5 which presents a (non exhaustive) list of DSG characteristic depending on several domains of Informatics. Agent and Grid are precisely described. Related work concerned with the shift from PD system to DSG systems is cited. Finally, sections 6 and 7 round off the paper with a discussion on the perspectives and the conclusions.

2 The concept of service

2.1 Theories and definitions

Humans beings need to help each other, they need to accomplish many tasks together. One may say that the more animals cooperate, the best their societies perform and evolve. Needless to say that cooperation in humans has been boosted by the emergence of languages, written languages, from the press to Internet, i.e., communication means. Since the whole information and knowledge, cannot be wrapped in only one man's hand, collaboration is crucial for information and knowledge sharing. The notion of service operates here: education, health, justice, commerce and trade are the results of services men provide one another. From a computing point of view, the end of the previous century has been marked by automation and computerization of various tasks and processes. Among them we distinguish service oriented scenarios. Nowadays, computing artefacts are able to replace our human fellows in several tasks such as: mathematical calculus (more quickly than with tables or abacuses), train tickets selling (without waiting in the ticket window line), money withdrawal (without even speaking to the banker), one click buying on the Web, one click googleing to get any kind of information that someone decide to publish somewhere on the Web, etc. Different interests (cultural, educational, economic, etc.) brought these services to be computerized in order to be more available, to concern more people, and to participate to a uniform development of people on the planet. However, could we really talk of *services*? What is a service? What differentiates the delivery of services from the one of simple products deliveries?

2.1.1 Dictionary definitions

Etymologically the word "service" comes from the Latin *servitium* (slavery, servitude), and from *servus* (slave) (see serve) and means the "act of serving". But real services involve people (or communities) sharing knowledge and know-how; they are not concerned (except in some usually service expressions such as "à votre service") by subordination of one on another. According to the dictionary, the term service is most of the time associated to an adjective and this pair carries the meaning. Thus, one will speak of national service, public service, weather service, community service, customer service, health service, postal service, information service, military service, transportation service, table service etc. The word is also often prefixed by a verb that specifies the service delivery action: services may be offered, provided, invoked, fulfilled, delivered, performed, exchanged, rendered, built, achieved, realised etc. The idea of service is so pervasive that we are used to say that we live in a "service society". The elements of the service exchange are themselves called: quality of service, terms of service, condition of service, denial of service, service contracts, service actors etc. These expressions indicate a general definition of service that could be: *use that one can make of something, someone.* The expressions *in/out of service* are used for a functional device or person exercising his/her role (or not). Actually, this definition assumes the pre-existence of a tool/person that should be used. The service is therefore associated to a functionality, a role. However, in the dictionary, another definition is more relevant: *what one makes for someone or work done by one person or group that benefits another or the contribution of an answer to the problem of another*. It defines neither a server nor a user but the entities accomplishing some task one another. In this second definition, the service is seen like a kind of collaboration/cooperation. In economy, the service, called intangible/immaterial good, is a good produced and consumed at the same time; it is viewed as a transformation. We should also note that a service can also be qualified of continuous, interrupted, intermittent, periodic, temporary, etc. These qualifiers, as the previous past participles, indicate a service does not exist (or pre-exist) alone; it is inevitably associated to a process achieving an action. It is important to notice that this process should answer a need or a wish. To sum up, the notion of service should be seen like a relationship between two (or several) entities whose goal is to solve the problem of some among them. The user of the service should be able to specify this action which must be created and adapted to his/her needs. The key element of the service notion is not therefore the product/good resulting from the service but the relationship created through service exchange interactions. We will see how DSG tries to define this kind of service in computer-mediated contexts.

From these first definitions we extract three first ideas that characterise the concept of service and that we keep in mind in the subsequent DSG description:

- Services imply creation of something new;
- Services are associated with processes;
- Services are constructed by means of conversations.

2.1.2 Economic definitions

Economists are used to talk about a ”service economy”, usually contrasted with a ”goods-producing economy”. They refer to an economy based on service exchanges rather than (manufactured or physical) goods. We propose in this section to connect our terms to an existing taxonomy: the economic one. In economics, the triplet good/service/product are the terms defining manufacturing exchanges (sales) between economical actors [2]:

- A (manufactured) good means a physical, tangible object (natural or man-made) used to satisfy people's identified wants and needs and that upon consumption, increases utility. It can be sold at a price in a market (if the purchaser considers the utility of the object more valuable than the money).
- A service is an activity that provides direct satisfaction of wishes and needs without the production of a tangible product or good (i.e., non-material equivalent of a good). It is claimed to be a process that creates benefits by facilitating either a change in users, a change in their physical possessions, or a change in their intangible assets. Examples include information, entertainment, healthcare and education.
- A product is a generic term for a tangible good, or an intangible service. This is the output of any production process. A product is anything that can be offered to a market that might satisfy an identified want or need. It is the complete bundle of benefits or satisfactions that buyers perceive they will obtain if they purchase the product.

Following the economic definitions, information processing is systematically considered as a service. Indeed, services that computers may provide are by definition virtual and do not have tangible/physical form e.g., information retrieval, weather forecast, calculus, communication tools etc. Even if services produce ”physical products,” e.g., a book purchased on the Web, they are systematically considered as service (e.g. we talk about the Amazon e-commerce service). This paper claims that the economic definitions, adapted to a world where physical abstractions exist have limited applications in computer-mediated contexts. In Informatics[3], a first differentiation is done in services by considering ”non virtual services”, i.e. services in non computer-mediated contexts (e.g. school) and ”virtual services” i.e., services in computer-mediated contexts (e.g. e-learning). We propose in this paper to go further by giving a fine grain differentiation of virtual services (cf. figure 1):

- product deliveries (PD), which are basic virtual goods produced by a pre-determined mechanism answering a parameterized question; akin to function.
- dynamic service generation (DSG), which are complex virtual services based on a conversation between user and provider which aims to resolve a need/problem of one of them.

We will further study the differences between product deliveries and dynamic service generation in section 4.3. However, we may immediately remark that product deliveries are associated to the result of an activity, highlighting the idea of obtaining the delivery of something already existing, or already designed, whereas the notion of dynamic service generation highlights the idea of process and the creation of something new.

The term production applies both for product and for service, and puts in evidence the idea of process, i.e., a composition of activities: *the act or process of producing something*. A production is the set of operations that allow one to get, by combining and composing existing products, indirectly usable, a new (manufactured or not) good/product adapted to fulfil a specific need. Production was firstly (19th century) limited to the process of creation (e.g., agriculture) but was extended/spread to the idea of transformation (Say and Malthus' approaches). In order to detail the production process, we will use the term *generation*, i.e., act of bringing something into being when talking about service (or DSG), and the term *delivery*, i.e., the act of delivering or distributing something (as goods) when talking about product (or PD).

Concerning service actors, the terms client and seller fit well with commercial situation. The terms teacher and students fit well with (e)-learning situation. We will use the generic terms *service user* and *service provider* to identify respectively the agent which benefits from the service, for whom the service is realized and the agent that supplies the service.

[2] (from Wikipedia, the free encyclopedia (`www.wikipedia.org`) and the AmosWEB Economic GLOSS*arama (`www.amosweb.com`)

[3] We rather prefer the term ”informatics” to the term ”computer science”. As the School of Informatics at the University of Edinburgh defines, it is the study of the structure, behaviour, and interactions of natural and engineered computational systems (i.e. representation, processing and communication of information in natural and artificial systems.)

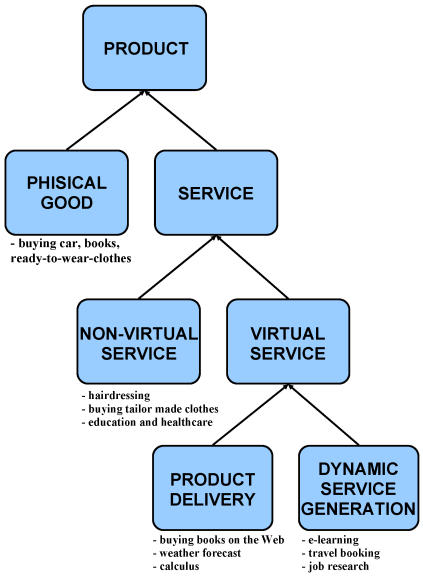

Figure 1: Economic taxonomy extension

2.1.3 Nifle's active appropriation services

Nifle, in his sociological and philosophical papers [Nif04a, Nif04b], reflects on the meaning of service. In [Nif04a], he proposes a service typology based on four aspects:

Assistance. The service is viewed as a presence by the one(s) that have a problem. It supposes a monitoring; the service provider accompanies the service user although without taking care completely of his/her problem at his/her place.

Substitution. The service is viewed as complete taking care of the service user problem by the service provider at his/her place. The problem is reduced to its objective independently of the user.

Mastery. The service is viewed as the exercise of a work or profession characterised by a mastery in a problem domain. It is an answer to a problem by the service provider in a problem domain on which the service user lacks mastery.

Subcontract. The service is viewed as an answer to a precise (and imperial) need. In this aspect, there is a relation of servitude as the need has to be satisfied.

As shown by figure 2 (simplified version of Nifle typology [Nif04a]) the combination of these aspects make two classes of services emerge. The first one, *active appropriation services*, consist in improving user's mastery of his/her problem. The service provider should help the user to progress in the resolution of his/her problem. The pedagogical dimension of this kind of service is very important as well as the way the user is taken into account. This kind of progress is similar to Socrates' Maieutics[4]. The problem is related to the service user (as the user expressed it), the service performance is related to the service provider (as the provider previously described it) and the solution is related to and benefits both of them. The second type, *administrated services*, consists in delivering a pre-composed solution. There is no dialogue, and no user adaptation or consideration; there are only abstract needs and concrete impersonal and standard solutions. The service provider is viewed as a specialist who demonstrates his know-how in obtaining results, by applying cognitive or technical procedures. The objective is absolutely not the improvement of the user mastery.

In [Nif04b] Nifle asks the question: Which concept of service is suitable for representing the Internet phenomenon? For him, Internet deals with relationships. Information and communication are the means. Of course

[4]It is a method of teaching introduced by Socrates based on the idea that the truth is latent in the mind of every human being due to his innate reason but has to be "given birth" by questions asked by the teacher and answers given by the student.

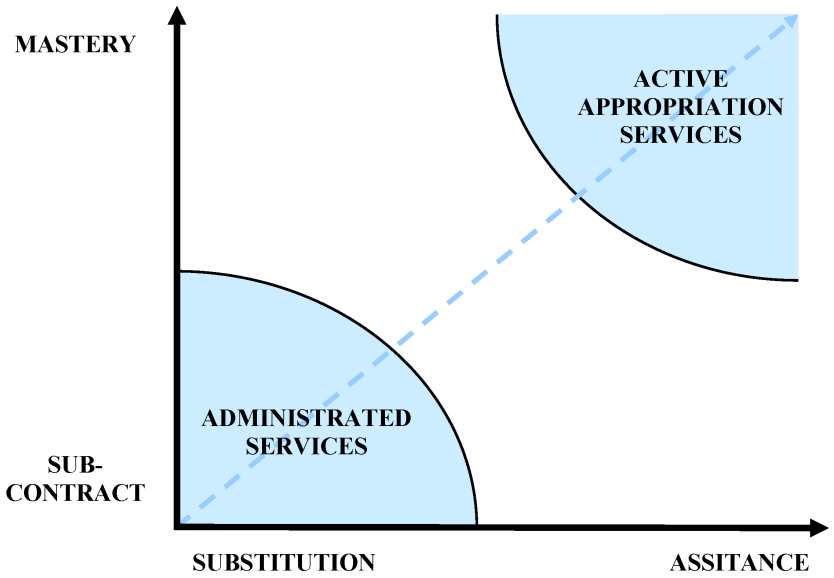

Figure 2: Nifle's service topology

Internet conveys information but it is not its fundamental role (intrinsic meaning). The main interest of Internet is to help to establish remotely close relationships. Besides, Internet is not a new medium because it fundamentally differs from classical (passive) media. All services based on a passive consumption are doomed to failure. Through Internet, users are seated in a driver position, they are active. Moreover, Internet is based on a community paradigm, it is absolutely not individualist. Users participate in multiples communities and emancipate in virtual worlds. Therefore, Nifle's suitable services for the Internet are those which facilitate relational games (which are inscribed and ease human relationships) within virtual communities. Active appropriation services are suitable for Internet.

From Nifle's reflection we extract two other aspects of the concept of service that we also find important for DSG description:

- Services aim to improve mastery;
- Services create relationships between members of communities.

3 Computing with services

3.1 Service oriented architecture principles

3.1.1 Computerization of the concept of service

The concept of dynamically generated services is not new outside the Informatics domain. We can say that since millennia, human beings can[5] generate "dynamically" one another many "services". It is intrinsic to humans. Humans are able to understand the hidden meaning of a sentence/expression, a feeling expressed by facial reactions, to recognize some shapes, some analogies between concepts, objects, needs, to infer some behaviours, to learn according to experience. Computers are not capable of all of this. Thus, what is so difficult in the computerization of the concept of service?

The word "services" seems to be the buzzword of our era. Expressions such as "service oriented architecture" (SOA), "service oriented computing" (SOC), "service based applications" (SBA) show the importance of the concept of service in the development of distributed systems. They define a model for the execution of distributed software application. This is historically due, for example, to the work of the Open Group with Distributed

[5]Of course, if they really want it: consider the real problem of a shared goal between provider and user, e.g., remember the failure in Clancey's journey example.

Computing Environment (DCE) and the Object Management Group with CORBA or Microsoft with (Distributed) Component Object Model (COM/DCOM) or Sun with Java Remote Method Invocation (RMI). The first ideas of these software component based approaches were i) to standardize invocation mechanisms; ii) to make transparent the location of these components on the network. Then, from the success of XML languages, the concept of service detached from a common middleware emerged, a notion based only on standardized and interoperable protocols and technologies over the Internet.

In computer-mediated contexts, a service is neither software nor process, but an abstraction (such as it is explained in [ASS96]) that implements a functionality. A service performance is done by message exchange. SOAs are said to be loosely coupled, that means that they are implementation independent, they do not have technological constraints, the failure of a part of the system doest not imply a total failure of the service, communication is done by asynchronous message passing etc. A SOA is said to be dynamic when it is not completely pre configured during the development and spreading phases and some parts of the service is specified and (re)configured at run time (on the fly). Dynamic SOAs are based on conversations that allow service providers to dynamically change and adapt the service by communicating. Loose coupling is naturally more dynamic than strong coupling.

For Singh and Huhns [SH05] the benefits of SOC are both that: i) SOC enables new kinds of flexible business applications of open systems that simply would not be possible otherwise; ii) SOC improves the productivity of programming and administering applications in open systems. The next sections detail some aspects of SOA.

Message passing based communication In SOA, methods of the services are not directly invoked by users, but when messages are received, *a service provider decides how to proceed by interpreting the user's message and invoking itself the good method(s)*. This is an important difference with software component based approaches. This is called message passing based communication, and it was originally suggested by Hewitt [Hew77]. The traditional formal model for communication is the Shannon statistical communication theory[6]. However, this popular model is very limited because it is basically a statistical model of information transmission. In real communicative situations such the ones that may occur in service exchange scenarios, it is impossible to consider communication as a simple information transmission, all the conversation or dialogue should be taken into account. This is especially addressed by agent communication research such it is presented in section 5.2.3.

Message passing communication can be *synchronous*: a rendezvous between communicative entities is needed; or *asynchronous*: a message can be buffered and an entity can send a message to another one even if the second one does not want to receive it or does not know about it.

The service contract A service is often described by a contract [MBG03]. A contract describes the reciprocal engagements of the service user and the service provider. It formalizes the service exchange: a service performance results from the execution of a contract. The elements of the contract are said to be "produced" and "consumed". These elements are [MBG03]:

1. The *service agent' identification* contains at least the identify elements (e.g. a X509 certificate, a Uniform Resource Identifier) of the service user, the service provider, but eventually identify elements of other agents such as third parties or a mediator. All these agents may be of course human (that interact via a specific interface) or artificial: a service may be realised for and by any kind of agents.

2. The contract uses a *service functional description* model which describes concrete service capabilities. It is called the service functional model and contains the specification of the service objectives, actions realised, information needed, rules applied etc. This model is of course not an implementation model in order to avoid that it is too strongly related to the service itself and that it reveals some information about the way the service is realised.

3. A service is often seen as a black box abstraction that interacts with users by message passing via an interface. The *service interface specification* contains therefore the message format description, and the message exchange protocols as well as the syntactic and eventually semantic description of the message elements. It also contains the definitions of bindings which formalise the relation between the service capabilities and the

[6] It considers four entities: a sender sends a message on a communication channel to a receiver. This message is written in a communication language. This simple model is the foundation of all communication formalisms. The transmission is considered as completely reliable if: i) Messages are emitted once or not at all; ii) Messages and emission/reception states are persistent and durable. To assure this reliability, a communication mechanism should provide: i) A message repetition mechanism (coupled with management mechanism for replicas) in case of channel failures; ii) A transaction management mechanism in case of failures of the sender or receiver.

protocols and messages. Interaction with the service interface should be for example: simple message sending, request/answer messages (the main one), sequence (streaming) of messages, multiple request/answer messages.

4. The *service operational description* corresponds to the quality of service specifications. The quality of service is a set of operational properties that should be verified during the service performance. These properties are:

   - The service domain of application, with the limits of the performance, rights and obligations of agents implied in the service exchange, but also the conformity to standards, etc.
   - The service quality of performance such as dimensionality, effectiveness, accessibility, accuracy and precision of the service.
   - The service security conditions and level with authentification policies, authorization listings, privacy rules, integrity etc.
   - The service robustness description with reliability, availability, continuity of the service, but also some technical properties such as transactions management[7] etc.
   - The service management description which specifies who drive, and monitor, the service performance. What will be the follow-up, the warranty etc.
   - The rules and adaptation to changes such as dysfunction, evolution, versioning, etc.

5. The *service contract life cycle* details the service performance lifetime as well as the eventual service performance frequency, the conditions of renewal, the conditions of service end over, etc.

6. The *service exchange description* specifies some properties about the service. For example if it is free or not. If not, which is the payment condition and whether it id an inclusive price or a performance unit price.

Service creation and service performance There are three ways for creating a new service: i) the *virtualisation* of an application (i.e. the "servicisation") which consists by taking an already existing application and transforming it in a service. For example, in SOA, interfacing a private application with the SOA standards; ii) the *service dissemination* which consists of creating a new service that directly realises the wanted functionality from scratch. For example, in SOA, writing a new SOA compliant service; iii) the *service aggregation* which consists of creating a new service by aggregating, composing already existent services (in SOA, for example writing a new business process).

Service performance may be of three types [MBG03]:

Stateless performance. No state changes during the performance. The service provider executes some tasks that produce information results directly transferred to the user. Services are also said to be stateless if they delegate responsibility for the management of the state to another component. A stateless Web service can be restarted following a failure without concern for its history of prior interactions. The same occurrences of the service should be executed several times without other effects than resource allocation. When the service is re-executed, the information that it delivers may be either invariant or variable, however the underlying application should manage evolving states. Take for example a clock service or a weather forecast service.

Internal stateful performance. The service provider executes some state transitions on its set of data (ex: Data Base Management System service). A stateful service has internal states that persist over multiple interactions. Stateful services should keep a state and evolve not simply with external messages (as simple objects) but also according to their own state. The same occurrences of the service produce successive state transitions. These internal state transitions are in principle reversible. Take for example a ticket reservation service.

Resource stateful performance. The service provider executes some state transitions on an external resource (environment) or agent (including the service user) consistent with internal state transitions. These changes are called side effects (based on assignments) and are by nature irreversible. Take for example a ticket buying service.

[7] A *transaction* is a process (i.e., program in execution) that accesses and possibly modifies data. Transaction processing mechanisms ensure that all or none of the steps of the process are completed and that the system reaches a consistent state (ACID properties: Atomicity, Consistency, Isolation and Durability).

The real challenge is to preserve stateless service qualities and facilities without storing a static state at the service level but rather change a dynamic state provided by another component with which the service interacts. That is the idea of WSRF presented in section 5.2.5.

Business processes management Business Process Management (BPM) is the term which groups all the approaches promoting high level processes of service: business processes, composition, aggregation, orchestration, choreography, workflow, conversation etc. These terms have to be further explained.

In SOA, a business process is a structure that defines logical and temporal relations between services. It is a service aggregation[8]: the aggregate service performance is the result of the combination of the aggregated service performances participating/engaged in the process. In some aspect BPM is similar to the planning problem that has been investigated extensively in AI [AHT90]. However two assumptions are often made in classical planning approaches: the world is static except for the actions of the planner, and the planner is omniscient. These assumptions are not valid in SOA. A business process design needs a cooperation/coordination between composite services participating in the process. This coordination and cooperation of the tasks needs imperatively communication between composite services and may be static (i.e., done before the business process execution) or dynamic (i.e., realised at the same time as the business process is executed). Thus, the aggregation of services in business process is often described under the terms of respectively conversation/choreography and orchestration/workflow of Web services.

Orchestration defines interactions and their sequences (described in term of messages and their interpretation) between composite services participating to the business process; akin to a "workflow" (as BPEL4WS). The description of this static composition of services is local and private to each participant to the process. The choreography connect orchestrations one another. Choreography specifies message exchanges as well as the sequence of these exchanges; akin to a "conversation" (as WSCL or WSCI). It is a participant common and public description (therefore the language of choreography must be standardized while the one of orchestration does not necessarily).

Service registries In order for a service to be used, it needs to be discovered by a user and a correspondence established between the goals of the user and the capabilities of the service. Service selection depends on reputations mechanisms, recommender techniques, referrals, trust. Let us consider for example three cases in user/provider meeting:

1. No third party, no mediator, the service provider and the service user know each other; the service performance is normally executed.
2. Intervention of at least one third party: an asymmetric registry. The service provider publishes a service offer in a services registry and the service user find this offer via the registry.
3. The same as (2), but both the service provider and the service user publish respectively an offer and a demand in a symmetric registry. The role of the registry is more important as it has to fit offers with demands.

These three cases show the different roles a registry should have. The purpose of a directory service is to enable service providers and users to locate each other. Most of the time registries are simple service directories in which service descriptions are available for potential users. They are called asymmetric registries. Classical functionalities of these registries are yellow pages (mechanism to find service providers by their characteristics and capabilities according to a standard taxonomy of domains), white pages (mechanism to found service providers by their identity (id?) (e.g., names) and green pages (mechanism to find all the services provided for a given service provider). These basic directories are simple databases with a passive behaviour.

A more advanced directory might be more active by providing not only search service. In these registries, offer and demand are published respectively by providers and users. Several offers for one demand or the opposite may exist. The role of the registry is therefore more active and consists by mapping these offers and demands together. This mapping could be realised by negotiation protocols e.g. auctions, RFP (Request for Proposal), CNP (Contract Net Protocol) etc. The registry plays the role of market place, and proposes a brokering or facilitating service added to simple search service. This kind of registry may be very useful for the services valuation as the registry may compare offers and demands.

[8] We prefer the term aggregation instead of composition as a business process is not a simple composition i.e. a series of service invocations in pipeline (where the output of a service is the input of another one) or a set of recursive RPC. In the following, we will use them in the same sense: composition refers to any form of putting services together to achieve some desired functionality.

3.1.2 Taxonomy of services

This section proposes a brief overview of SOA aspects as shown by figure 3. It helps us to identify and classify services according to a set of 11 questions/criteria:

- Does the service communicate by asynchronous or synchronous message passing?
- Does the service manage an internal persistent state?
- Is the service accessible via a system oriented architecture or a service oriented architecture? (if not, we should better talk about software components)
- Is the service compliant with SOA standards? Is it possible to publish it as a standard service?
- Does the service manage its lifetime? Is it transient or persistent?
- Is the service a collaborative one? Is it inscribed in a collaborative (business) process implying several users and providers?
- Is the service defined by a contract?
- Is the service execution a simple one shot interaction (e.g. request/answer) or a long lived conversation?
- Does the service have a semantic description? Is it able to deal with semantics and ontologies?
- Does the service usage imply only human or artificial users? Is it able to provide the functionality to any type of user? Is the service itself provided by a human or artificial service provider?
- Is the service a simple one or an aggregated or composed one?

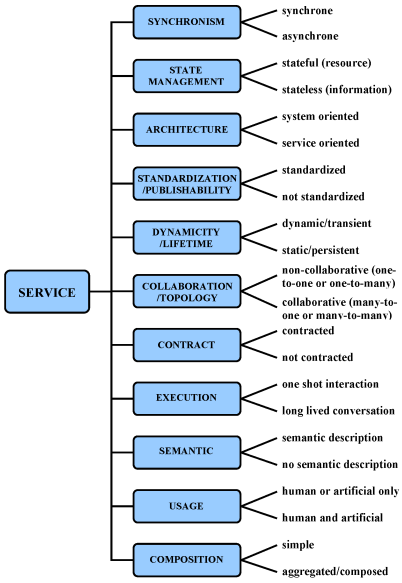

Figure 3: Service properties

3.2 Service oriented architecture standards and technologies

3.2.1 Web service

Nowadays, the main way of implementing a SOA (framework) is by means of Web services (`www.w3.org/2002/ws`). The identification of the concept is sometime attributed to D. Winer who proposed in the beginning of 1998 XML-RPC as a RPC mechanism based on XML (UserLand `www.xmlrpc.com/spec`). Actually, as the W3C defines it, a Web service is: *a software system identified by a URI (Uniform Resource Identifier), whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols*. Web services are describable, discoverable and message based software components that perform some function. Many Web service technologies exist (cf. figure 5) but Web services are mainly based on the three XML based standard languages: i) WSDL (Web Services Description Language) to describe software components i.e. functions that may be invoked; ii) SOAP (Simple Object Access Protocol) to describe methods for accessing these components i.e. message exchanges; iii) UDDI (Universal Description, Discovery and Integration) to publish a service and to identify/discover a service provider in a service registry. In Web services, providers publish their services on registries, and consumers find the service providers from registries and then invoke them such as illustrated by figure 4.

There are two main views of Web services: the *RPC-centric view* (Sun) treats services as offering a set of methods to be invoked remotely and the *document-centric view* (Microsoft and Sun) which considers the documents as the main representations and purpose of the distributed computation. Each component reads, produces, stores, and transmits documents.

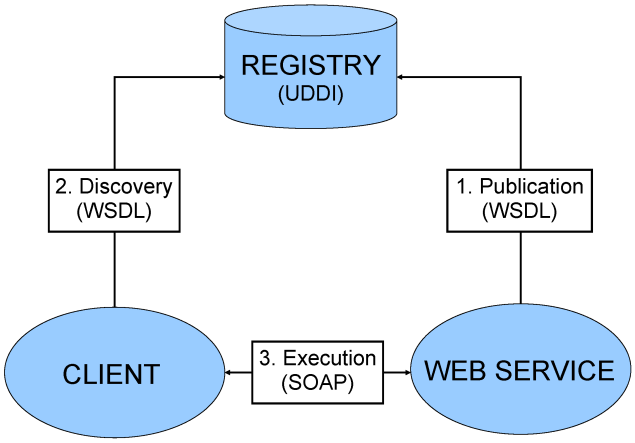

Figure 4: Web service life cycle

The two main objectives of Web services are standardisation and interoperation (intra-enterprise and inter-enterprise). Web services allow to access distributed functionalities on a network in a standardised way. With Web services, different applications can communicate (through HTTP and firewalls) with each other without knowing anything special about their implementation (operating system or programming language) but only dealing with a standardised interface where exchanges are expressed by XML documents. Standardisation enables both services to be accessed by any kind of agent and interoperability between these services. Certainly, the main advantage of Web services arises when we can compose them to create new services. Thus, from service invocations, which are single-shot two-party interactions, Web services started to evolve to business processes that are typically long-lived multiparty interactions. This is called Business Process Management (BPM); from this interest, another set of languages, called Process Description Language (PDL), emerged.

The two main Web service development platforms are J2EE (from Sun and JCP (Java Community Process)) and .Net (from Microsoft). They are able to generate WSDL and SOAP standard documents from platform dependent languages (Java, C#).

3.2.2 Web services limits

Simple Web services have some important drawbacks such as: RPC like computing, object-oriented behaviour, client/server oriented, no user adaptation, no memory (stateless), no life time management, no conversation handling (simple request/answer interaction). SOAP is a stateless protocol and each SOAP message is unrelated to any other message. Hence SOAP does not describe bidirectional or multiparty interactions. One can use conversation identifiers at the application level to build a conversation. Web services are passive (non proactive) and they do not take into account the autonomy of components, neither they use a high level, history aware, communication language. Furthermore, a major drawback of Web services is semantics: WSDL allows definition of a service interface and service implementation but only at a syntactic level. There is no way to transfer transaction semantics across a SOAP call. There are two major standards for directories: ebXML registries and UDDI registries (themselves described and interfaced as Web services). Unfortunately, neither supports semantic descriptions/searching of/on functionality. Searches can only be based on keywords such as service provider name, location, or business category. ebXML registries have an advantage over UDDI registries in that they allow SQL-based queries on keywords. Indeed, none of them provide high level discovery and selection mechanism. Therefore, they do not play the role of a market place.

[HNL02] states that Web services remain a ”vending machine” model, what we call later: the product delivery approach, in opposition with a dynamic service generation approach. All these aspects and drawbacks imply that the Web service framework as it is now does not fulfil DSG requirements. Actually, for us, Web services are typical PDS: remote parameterized and standardized functions accessible by RPC. However, some ideas coming from this framework are very important and are the origin of some DSG characteristics cf. section 5.2.2.

3.2.3 Web services extension: Semantic and Grid systems

The lack of semantics of Web services is at the origin of research on ontologies, Semantic Web and Semantic Web Services i.e., semantically described Web services. The two main current technologies are proposed by the OWL-S (OWL-based Web Service Ontology) and WSMO (Web Service Modeling Ontology) working groups.

The recent interest about Grid systems [FK99] is at the origin of research on Grid services [FKNT02] as compliant SOA standard services available through Grid infrastructures (Grid systems, Grid services and the major progress for SOA are precisely described in section 5.2.5) with two main technologies: OGSA (the architecture) and WSRF (the infrastructure).

3.2.4 Business Process Management technologies

As it is a quite recent interest, there is no real standard for the moment in BPM technologies and PDLs. Each consortium and/or company proposes its own framework (see Appendix A). We mention here three of them. On the workflow/orchestration side, BPEL4WS (Business Process Execution Language for Web Services) (coming historically from IBM and BEA WSFL (Web Services Flow Language) and Microsoft XLang) seems to become the de facto standard. Structurally, a BPEL4WS process describes a workflow by stating who the participants are, what services they must implement in order to belong to the workflow, and what are the various orders in which the events must occur. That is, a BPEL4WS process describes the orchestration of a set of messages all of which are described by their WSDL definitions [BVV03]. BPEL4WS allow to express precedence constraints more complex than those expressed with UML. BPEL4WS uses WSDL to model both the process and the participating Web services. Therefore, BPEL4WS business process is described as a new Web service that could be implied in other business processes.

On the conversation/choreography side, WSCL (Web Services Conversation Language) specifies the sequencing of XML documents – as well as specifications for the documents themselves - being exchanged between a Web service and a user of that service. A WSCL conversation is a XML document. WSCI (Web Service Choreography Interface) provides a global message-oriented view of the choreographed interactions among a collection of Web services. It describes the flow of messages exchanged by a Web service that is interacting with other services according to a choreographed pattern (protocol). A WSCI specification is part of a WSDL document.

The main drawback of these approaches is dynamicity. Most of the time the business process or composite service is designed before its execution: it is precompiled and ready to be triggered. Even if the workflow engine can execute the workflow orchestration invocations asynchronously, the process is still centralized, which means that it suffers from the single point-of-failure weaknesses that characterize centralized systems. These are the important aspects that agent technologies may improve. Section 5.2.3 explains how this evolution from simple

service exchange to high level processes of services could be done only by substituting the object oriented kernel of service oriented applications by an agent oriented one able to have and manage conversations to realise dynamically generated services.

Figure 5 tries to sum-up the current SOA technologies and standards. See appendix A for a description of acronyms and URLs.

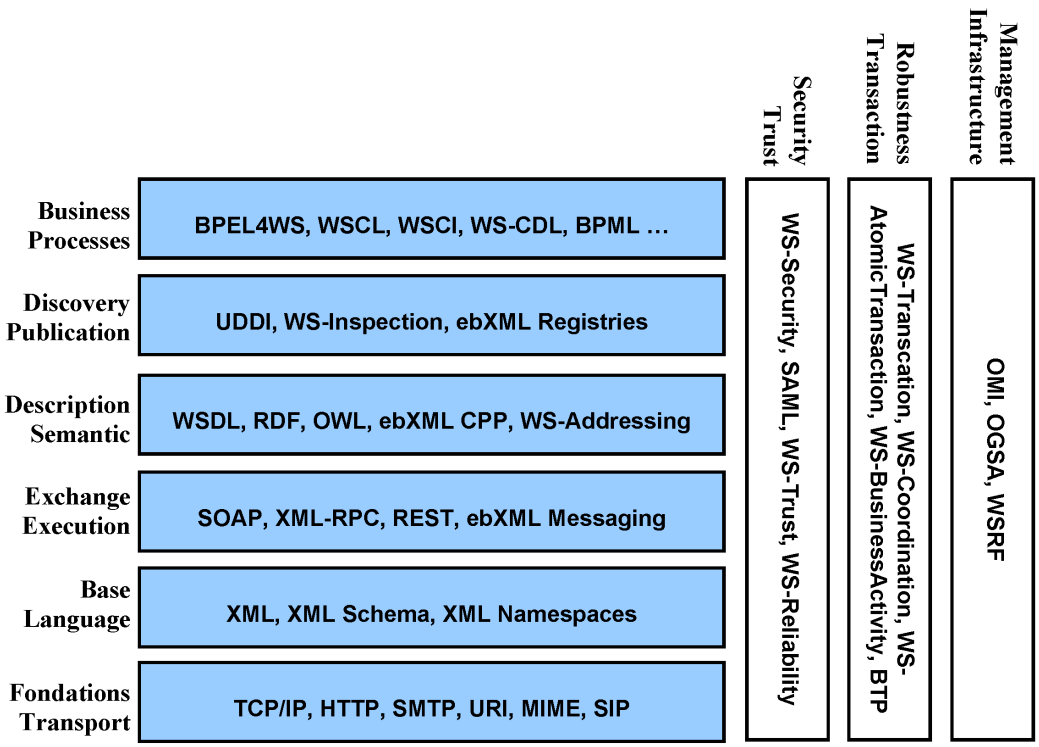

Figure 5: Service oriented architecture technologies and standards diagram

4 Dynamic Service Generation

4.1 DSG examples

In order to have a real image of what is DSG, let us consider some examples. As explained in the beginning of the paper, human beings generate services each other since always, so let us start with scenarios external to Informatics and continue with more computer oriented examples.

4.1.1 Human-centered service scenarios

A travel/journey planning requires a dynamically generated service, most of the time accomplished by a travel agency, allowing a user to express his/her real objectives, possible constraints on dates, his/her centre of interests, the preferred transportation, priorities, family/professional constraints etc. in order to construct a special travel/journey. It is much more than simple ticket buying, as it is evident for example, from the scenario reported by Clancey in [Cla05].[9]

When a client goes to a train ticket automatic machine (i.e. a PDS) he/she knows exactly what he/she wants and expects to obtain quickly the product. Sometimes, clients wait half an hour on the train ticket selling window line because they know that the automaton would fail in satisfying their request, or would not be able to answer a specific question. They want to talk to a person. They want a service to be generated by this person.

[9]Notice that the travel planning scenario is typically presented in SOC when dealing with process management, Web service composition etc.

In general, in everyday life, it is not difficult to make a difference between PD and DSG. However, these examples show that the two approaches are not opposed but complementary. For example, we both need doctors and drugstore, train ticket selling automatons may be very convenient to win time when users know exactly what they want; some ready-to-wear clothes are wonderful, etc.

4.1.2 Computer-centered service scenarios

Of course, some of the previous examples may be viewed as future computing scenario (travel planning, job search, train ticket buying etc.). More generally, most domains of Informatics (agent and multi-agent systems, service oriented computing, e-learning, collaboration, distributed systems etc.) that deal with the concept of service will benefit from DSG.

A concrete example of computer systems that will benefit from the progress in DSG is operating systems (OS). An OS must be user adapted and must generate knowledge about how to use it. A user should understand progressively what the OS allows and discover the type of activity he/she may realise with the underlying computer. A user should have a trust relationship with his/her OS, he/she should be able to delegate tasks to it; the OS should be able to infer user needs according to the way he/she use and used this OS.

In some aspects, DSG is analogue to information retrieval with the following properties: i) the database is so large that users cannot specify a single, simple database query for which there is one single answer; ii) users may not necessarily have a definitive set of constraints in mind when starting the search task, or may change constraints during the message exchange, and iii) there may be multiple search goals; what satisfies a user will not satisfy another one.

DSG could as well be a base for development of future Intelligent Tutoring Systems (ITS) in e-learning. For the moment ITSs are most of the time limited to information transfer: the ”tutor” teaches what it knows to the learner. There is no real interaction (i.e. changes of state) between them: the tutor learns hardly anything. DSG should support exchanges in ITS and allow them to be more dynamic and thus to learn and create more knowledge available both for the tutor and the learner. This aspect is more precisely studied in section 5.3.3.

As a last example, we simply mention the progress actors such Amazon make when they introduce marketing concepts in their e-commerce Web sites. For instance, they propose on-line selling services more adapted to users and strategically thought in order for them to buy. The scope of dynamically generated services seems to satisfy the same need for adaptability in interactions within e-commerce scenarios.

4.2 Dynamic service generation elements

4.2.1 Agents and systems

We do not use the word service alone anymore to describe our concept of service, but we rather talk about **dynamic service generation** (DSG). This approach is opposed to the classical product approach, called **product delivery** (PD). The systems (i.e., computing systems), realised and used both by humans and computers, that provide dynamically generated services and delivered products are respectively called DSG systems (DSGS) and PD systems (PDS). They embody the role of the service provider. PDS produce one and only one well defined product whereas DSGS are able to generate many kinds of different, dynamically constructed services for human or artificial users. These systems do not exist for the moment. It is the main objective of the rest of the paper to propose a set of characteristics that will help to progressively shift from PDS to DSGS. Besides, some expressions related to these terms are also necessary. So, we will say that a user ”asks” or ”requests” a product from a PDS. A PDS will then ”deliver” a product. On the other hand, we will not say that a user asks or requests a service, but rather asks for a DSG or that a service is generated by a DSGS for a user.

PD and DSG imply two kinds of entities. The first one is most of the time called: human user, human being, natural entity etc. The second one is most of the time called: computer, machine, agent, software system etc. In order to uniform these two different kinds of user/provider, we will adopt a unified view of these entities as it is currently done within the agent research community. The agent concept [Fer99, Woo02, HS98] is for the moment (or tries to be) the best operational metaphor of humans in Informatics. In the rest of the paper, for the sake of simplicity, we consider two types of agents: **human agents** (HA) and **artificial agents** (AA).Three types of interaction occur in PD/DSG: AA-AA, AA-HA, HA-HA. A PD/DSG may occur provided by any kind of agent for a user of any kind. We use the term **community** to identify the social group in which agents evolve and generate services one another. Figure 6 summarizes DSG elements.

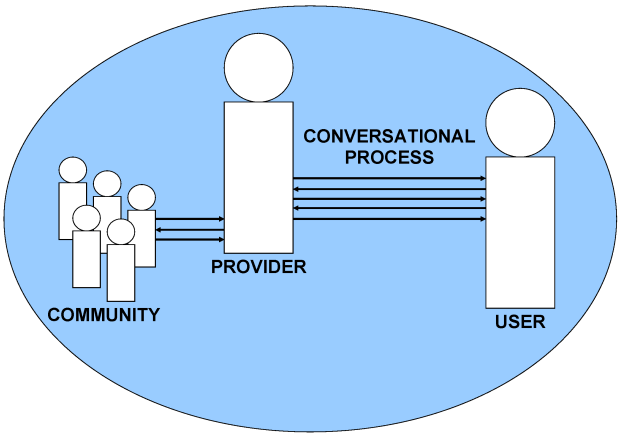

Figure 6: Dynamic service generation elements

4.2.2 Virtual community

A notion that appears in DSG is that of community. A PDS may be considered as a unique and independent system, alone and disconnected from other PDS, simply in relation with users. On the contrary, a DSGS may play its role only as a member of a virtual community, that means connected to other DSGS (or PDS) which allows it to dynamically construct the service it has to generate using its own capabilities and/or the ones of other member of the community (i.e. others agents of the community may generate dynamically services in order to help one of them to realise another generation of service).

These communities may be of little or large scale. They may be persistent for a long time or very ephemeral. These communities may be explicit (e.g., a working group) or implicit (e.g. all Web users). They are highly dynamic and evolve according to interactions and services generated. The notion of community is strongly influenced by Grid systems as presented in section 5.2.5.

4.2.3 Conversational processes

PD can be considered as the result of a one-shot interaction process between a pair: user - service provider. The concept of service is intrinsically bound to a generation process. This process is realised by a conversation between the provider and the user called the **conversational process** thus dynamically generated services may be viewed as the result of the activation and management of a unique process defined by the triplet: user - conversational process - service provider as shown in figure 7. The agents playing the roles of provider and user pre-exist to the DSG whereas the conversational process represent the conversation these two agents have to realise the dynamically generated service. A conversational process is a long lived interaction that allows user and provider to dynamically respectively express their need or wish and capabilities or constraints. Interaction plays a key/central role in DSG.

A process is defined as a naturally occurring (or designed) sequence of operations or events, which produces some outcome. A process may be identified by the changes it creates in the properties of one or more objects under its influence. In DSG, when the sequence of operations (i.e. messages) is designed, the conversational process is called a conversation. When the sequence of operations (i.e. messages) occurs naturally, the conversational process is called a dialogue (cf. section 5.2.3 for details on this concept of agent communication). Outcomes produced by conversational processes are the result of the DSG itself. Changes created on the object under the influence of the conversational process (i.e. agents) are changes of their internal states. Indeed, a conversational process is a set of interactions and an interaction between two entities implies an action to occur on the interacting entities. Thus DSG provokes changes of state; this is one of the reason that explains why each DSG is unique (changes of state are stored in a history).

The very beginning and the end of a conversational process implies the beginning and the end of the DSG. We talk about DSG **bootstrap** to identify the beginning of the conversational process. We call **elements**, the different changes provoked by interactions occurring during the conversational process. Elements are local to each agent implied in the conversational process. The end of the DSG is characterized, for each agent, by a **final result** as the integration of all process elements. This final result may be a physical or a virtual good, but also knowledge,

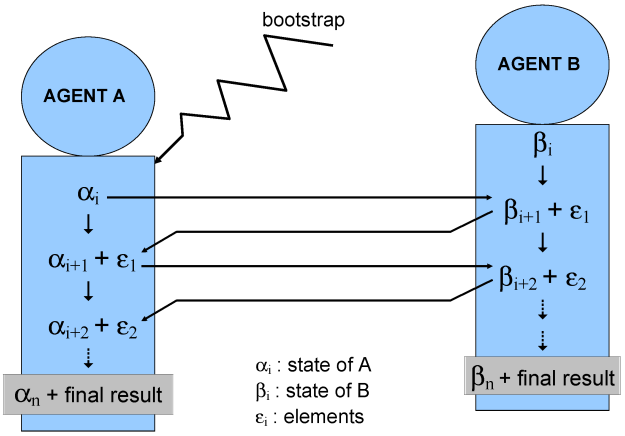

Figure 7: Conversational process

emotion, information, advice, qualification, or any kind of answer etc. The final result may be ”agent internal” (e.g. emotion, knowledge), ”agent external” (e.g. a physical/virtual good).

Table 1: Definitions of DSG elements

Concept	Definition
Dynamic service generation	The global process implying agents in a community (at least two: a user and a provider) in a conversation and producing a work which aims to solve a problem, or to answer a need or a wish (not necessary clearly specified) of one of the members.
Product delivery	The unique and final work, result of the activation of a functionality, or got by any activity, corresponding to a well specified need or want, produced for a user by a provider.
DSG system	Abstraction that represents a computing system able to realise dynamic service generation.
PD system	Abstraction that represents a computing system able to realise product delivery.
Agent	Computational metaphor to represent active entities (i.e., actors) implied in dynamic service generation or product delivery. There are two kinds of agents: human agents and artificial agents.
Provider	Name given to the agent that plays the role of the DSG system or PD system, which aim is to help a user to answer a need, want or wish.
User	Name given to the agent that plays the role of the requester of a product delivery or for whom a service is dynamically generated, by a provider.
Community	The social group (virtual or not) composed of almost two agents in which agents evolve by providing one another different dynamically generated services. Each member of a community may be both provider and user according to the situation.
Conversational process	Long lived interaction (i.e., conversation) occurring between a user and a provider during a dynamic service generation that allows user and provider to dynamically respectively express their need or wish and capabilities or constraints till the end of the generation of the service.

4.3 Dynamic Service Generation vs. Product Delivery

As told before, DSG is opposed to the classic PD approach. We outline here after some fundamental differences between PDS and DSGS; that permit to characterize DSGS a contrario (i.e. by opposition). Besides, these differences are not necessarily all and always true in DSG. For example, a user may know his needs (D1), but may not know the way to express them (D3).

D 1 (Need expression) *In PD the user exactly knows what he wants. His need is clearly specified. He knows which provider can help him to resolve his problem. For example, in SOA, a user discovers a Web service in a*

UDDI registry and invokes it according to its WSDL description (D3). In DSG, on the contrary, a user may want a service to be generated for him without knowing exactly what he wants; he can even be unaware of the fact that he needs a service to be generated (D17). The user is simply in a bootstrap situation, some of his needs appear during the generation process, other ones disappear.

D 2 (Offer expression) *In PD, the user knows what the system can offer him. He knows it because of previous PD, or recommendation, or advertisement, or precise description. Besides, a registry defines the provider available capabilities. In DSG the user elicits and understands what the service provider can offer him by interacting with it. He may discover some of his needs in the same time as he discovers the service provider capabilities (he discovers also other capabilities that would be used in future DSG (bootstrap)). On the other side, the service provider constructs something to offer to the user, according to the conversation it has with the user. In DSG, registries or any kind of index are difficult to set up as each dynamically generated service is unique and fits a very specific need.*

D 3 (Request expression) *In PD, the user knows how to express his request. He should adapt to the provider language. This is not the case in DSG. The service provider should help the user to express his request and try to understand his language. Note the important aspect of learning.*

D 4 (Uniqueness) *Several requests of the same PDS, by the same or another user, produce the same type of deliveries. Some products may be exactly the same if we consider the purely functional capability of stateless service provider. At the contrary, generated services can never be the same as they depend on a conversational process and a conversation is unique. Moreover, each DSG is unique also because of D5.*

D 5 (History) *Dynamically generated services depend one another; they are part of a history. A dynamically generated service will depend of course from the conversational process, but also from the current states of the agents, themselves changed by previous DSG. Three generated services in a special order would not have been the same in another order. PDSs do not have history; most of them do not have even a state.*

D 6 (Subsumtion) *PDS cannot realise DSG; PD may be composed of other PD but it is already difficult. On the other hand, DSGS subsume PDS, which means that a system capable to generate services can delivers products. Even if a DSGS only deliver products, it distinguishes PDS because of D4. A dynamically generated service can be the result of the aggregation/composition of other dynamically generated services but also of product deliveries. It is important to notice that the goal of DSGS is not to substitute PDS, but to be complementary. Very often, PDS perfectly answer the user needs and DSG is not necessary e.g., getting a phone number will always be a PD.*

D 7 (Development) *A PDS is developed by the provider (and is supposed to correspond with a well established and clearly identified need (a market)) with a clearly predefined goal for the potential user. At the contrary, a DSGS is offered within a service domain, so the user specific objectives have to be defined during the DSG conversational process.*

D 8 (Lifetime) *A PDS has a long lifetime as it is developed once by the provider and never changes. On the opposite, DSGS life cycles are ephemeral as they change and evolve with each generation of service.*

D 9 (Satisfaction) *The value added by a PDS increases with the number of products delivered. The value added by a DSGS increases proportionally with the users' satisfaction of the final result, which entails an indirect publicity (reputation) for the DSGS and thus stimulates new users ready to have similar services (bootstrap).*

D 10 (Determinism) *PD is a deterministic process that leads to a product. Computing principles behind PDS are deterministic: algorithms, execution etc. On the other hand, DSGS have non-deterministic behaviours; we should not really anticipate their final results. Computing principles should be non-deterministic (cf. section 5.3.2).*

D 11 (Retraction) *During a PD the user cannot change his mind. He cannot retract and must wait for the end of the delivery (generally fast) to express a new request with his new need or want. At the opposite during a DSG the user can change his mind at anytime during the process. Especially, he does that according to DSGS reactions.*

D 12 (Evolution) *A PDS evolution is slow, as it requires modifications in the conception, design and development - a revision of the whole life cycle. A DSGS evolves naturally as it is most of the time a on the fly combination of other dynamically generated services and PD.*

D 13 (Valuation) *The use of a PDS is easily valuable and billable. Since PDs can be the same (D4), the valuation (price) is calculated as a function of offer and demand. Besides, remote procedure calls executions may be anticipated and resources (time and space) should be previously reserved (algorithm complexity may be predetermined). Inversely, the use of a DSGS is neither easily valuable nor billable. Since generated services have interests only for a given user (at a precise time) and since they will never be regenerated in the same way, a function of offer and demand for helping to define the value is not easily to be found. It is impossible to really estimate how much a user should "pay" for a generated service. A dynamically generated service is also an investment of which it is very difficult to evaluate the potential return.*

D 14 (Psychology) *A PD does not provoke emotional reactions. The user knew what he wanted beforehand, no surprise or any other emotions are implied by the PD. Some emotions such as happiness, disappointment etc. may occur but they are related to the product received and not to the process of delivering this product. On the contrary, a dynamically generated service has a psychological effect on the agent implied; especially, it implies and provokes emotions (positive and negative). The dynamicity of the process (everything is not previously clearly defined) lets unforeseen events occur generating emotional reactions. These emotions play an important role on the agents DSG valuation (D13).*

D 15 (Predictability) *PDSs are able to announce the result of their use; they can "show" future results (i.e. obtained products). A PDS can for example explain situations before and after the PD. On the contrary, a DSGS must gain the trust/confidence of its users. It cannot announce a final result, nor guarantee it. Where a PDS cycle of use is defined by: estimation, order, delivery; a DSGS cycle of use is based on a trust delegation. A DSGS takes intelligent decisions for the user by/while inferring on its intentions. It must sometimes take the initiative (D17).*

D 16 (Reasoning) *PDSs are inactive when not engaged in a delivery phase. They only wait for next requests (as for example Web servers). At the opposite, a DSGS is never inactive; it is perpetually evolving, learning and reasoning on previous generations to improve the next ones. This evolution is based on the DSGS history (D5). A part of this reasoning is done dynamically (during service generations) and another part is done statically (after generations phases) for example analyze and change according to statistics. See also section 5.3.3.*

D 17 (Behaviour) *A PDS is passive, and never takes the initiative; it systematically waits to be called. It is able to publish and advertise its capabilities but not to start the delivery process. On the other hand, a DSGS is active and proactive; it may start a DSG process without the user's previous solicitation or agreement. It must be able to notice/detect bootstrap situations when they occur (i.e., potential users). Pro-activity should be appreciated if it is necessary (D15) and allows the DSGS to increase its value added. Besides, a user cannot appreciate a non desired solicitation. For these reasons, DSGS are good models of Ambient Intelligence.*

D 18 (Description level) *PDSs simply deal with the syntactic level (i.e. the basic interpretation of expressions) in their interactions with users. DSGS must also deal with semantic (i.e. meaning of expression) and pragmatic levels (i.e. the context of interpretation) within interactions, descriptions, etc.*

D 19 (Creativity) *A PD does not create knowledge. Users know what they want and there is no creation of new knowledge. Inversely, DSG generates knowledge. Merely, if D1 is verified, then users discover almost what they really want or do not want. See also section 5.3.3.*

The DSG approach does not aim to replace PD but rather to improve and enhance it. Generating a service means understand and construct, by interactions, what a user needs or wishes. If a user has precise needs or precise ideas of what he wants the system to give him, then interactions are simple and a PD may be rapidly achieved. However, many scenarios such as the ones presented in section 4.1 need the DSG approach.

5 A list of dynamic service generation characteristics

5.1 Methodology

The goal of this section is to extract from the paper a set of characteristics for DSG. This is not an exhaustive list, but it helps us to concretely represent the path for going toward DSGS. Actually, there is not a formal specification of DSGS, but there is a continuum of systems that start from nowadays PDS (Web services, multi-agent system,

distributed systems, Grid systems, Semantic Web Services etc.) and converge to tomorrow DSGS. The more a system achieves the characteristic of the list, the more it goes toward DSGS. [10]

5.2 Foundational domains of Informatics

We adhere to the scientific position taken by Singh and Huhns [SH05] that "it is not possible for computer scientists to develop an effective understanding of SOC by merely studying the basics standards for Web services but rather by examining several areas of Informatics that come together in connection with services. Many of the key techniques now being applied in building services and service based applications were developed in the areas of database, distributed computing, artificial intelligence, and multi-agent systems." Therefore, the first characteristics come from five research domains of Informatics that we further think are the foundation of DSG. These five domains themselves emerge from the distributed aspect computer science has taken in the last decades: i.e., distributed documents, resources, applications knowledge, and systems (cf. figure 8). Taken one by one, these domains present many interesting and promising qualities to realise future DSGS but of course, only a common development and integration of these domains will allow the emergence of real DSGS.

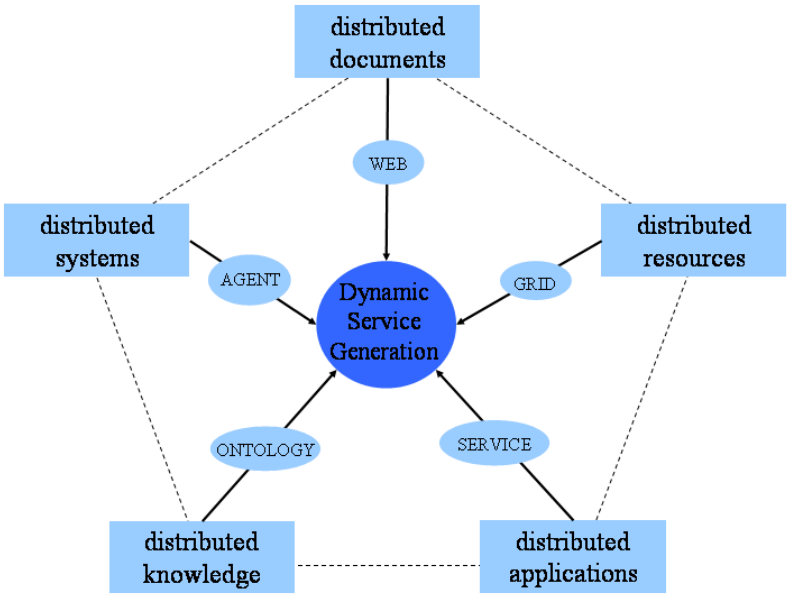

Figure 8: DSG as the integration of five domains of Informatics

5.2.1 Distributed documents: the Web

The World Wide Web is a network of servers linked together by common protocols, allowing access and exchange of millions of hypertext documents. The Web created a worldwide participatory movement toward a new world where everyone can start a relationship with everyone else, independently of the distance, following his initiative and creativity. Furthermore, the Web is an interaction space where agents in communities exchange information and knowledge. The challenge is to keep this statement true with service exchange on the Web. For the moment the Web is a non structured set of information designed for humans and lacks semantics. However, the Web is a key element of DSG. DSG would enable the Web to become a real collaboration space by allowing all kind of agents to generate services one another.

Characteristic 1

[10] Notice that some characteristics presented here are more detailed, which does not mean that they are more important, but rather they fit better our domain of competence. We are confident that more accurate description of these characteristics will naturally emerge from the different communities they are related to.

DSGS have to be Web oriented. They have to be accessible through Web standards and technologies but the Web also has to evolve to fit other DSGS requirements (e.g. statefulness, semantics etc.)

5.2.2 Distributed applications: Service Oriented Computing

Even if Web services is not an answer for DSG, we can extract from the analysis done section 3 some characteristics of SOC fundamental for DSG.

Characteristic 2

DSG should use an asynchronous message passing based communication.

Characteristic 3

DSGS should be able to address dynamic business processes (i.e., not precompiled ones) by communication, in a workflow based approach as well as in a conversation based approach.

Characteristic 4

DSG needs asymmetric dynamic registries where DSGS and service users publish respectively (what they know, or what can be published from their) offers and demands. These registries should play the role of market places.

We do not take a position concerning the fact that DSG should be described by a contract.

5.2.3 Distributed system: Agent and Multi-Agents System

Agent An agent (i.e. AA) [Fer99, Jen01] is a clearly identifiable physical or virtual autonomous entity which:

- is situated in a particular environment;
- is capable of perceiving (with sensors) and acting (with effectors) in that environment;
- is designed to fulfil a specific role;
- communicates directly with other agents;
- possesses its own state (and controls it) and skills;
- offers services (in the sense of particular problem solving capabilities);
- has a behaviour that tends to satisfy its objectives.

Traditionally, one may distinguish two main types of agents: i) reactive agents, that have little perception of their environment and react to the information they received (most of the time via indirect message passing through the environment and ii) cognitive agents that have generally a more precise (but always limited[11]) representation of their environment, some precise skills and knowledge and that answer to messages directly sent by other agents to reach their objectives. Agents are reactive (able to respond in a timely fashion to changes that occur in their environment) and proactive (able to opportunistically adopt goals and take the initiative), capable and efficient (able to solve problems and reach private objectives) and adaptive (able to learn and change following experiences).

Agents and multi-agent systems (MAS) were extensively studied in literature, for example [Fer99, Woo02, HS98]. Historically, the agent paradigm comes from the object one [GB99], but differs according to three major aspects: autonomy i.e., for example, the fact of being able to refuse to answer a message; intelligence i.e., the ability to change its own state alone, without message passing; interaction i.e., the faculty to communicate directly and asynchronously with their environment and other agents by means of direct or indirect message passing with a communication language independent from the content of the communication and from the internals of agents. Agents have a persistent state. They are also able to use and reconcile ontologies. All these properties are very interesting for providing services.

[11] The limited perception of an agent environment is fundamental, it guarantees the real distribution of a system, indeed, a system that could be observed, mastered, in a global manner by one single agent would not be distributed anymore.

Actually, Web services are most of the time an interface on object oriented programs (developed via J2EE or .NET) processed to produce a WSDL description and to become able to communicate with SOAP messages. Thus, services benefit of the powerful abstraction characteristic available with the object oriented paradigm, such as encapsulation, inheritance, message passing, etc. One of the crucial evolutions of DSGS concerns the substitution of an agent oriented kernel to the current object oriented kernel for Web services. This will have the fundamental advantage to ensure Web services to be proactive (autonomous), intelligent and really interactive (the 3 fundamental properties distinguishing agents from objects). Viewing service providers and requesters as agents is for us the best way to realise dynamically generated services, and high level processes (business, composition, orchestration etc.) of services. The persistence of conversations among service providers and requesters (dynamic dialogue rules, not just prefixed interaction protocols as it is the case now), as well as the run-time learning/reasoning of them (i.e. progressive evolution of the performance) could be reached only with agents.

Characteristic 5

DSGS have to be agent oriented for two reasons. Firstly because of the adapted properties of agent for providing (and consequently using) services. Secondly, because agents are the current best metaphor for humans in computing i.e. agents enable to use a unique model for HA and AA.

The term ”societies of agents”, highlights the importance of the social aspect in MAS. A MAS is not a simple set of agents put together in a common environment, but a real organisation with social rules and interactions allowing cooperation and collaboration to resolve problems that centralized systems (as intelligent as they can) would not have resolved alone [Fer99, Woo02]. These societies of agents emerge through agent communication.

Agent communication Simply grouping together several agents is not enough to form a MAS. It is communication between these agents that makes it. Communication in MAS is highly influenced by language philosophy and specially speech act theory [Sea69, Aus62]. Searle and Austin explained that communication is realised by speech acts expressing an illocutionary force. Speech acts, which can be viewed as the linguistic unit of communication, had been classed by performatives. These works, and the influence of mental state centred architectures (mentalist approaches) such as BDI (Believe Desire Intention)[12] [RG91]] originated famous agent communication languages (ACL) such as KQML (Knowledge Query and Manipulation Language) [LF97] and FIPA-ACL (Foundation for Intelligent Physical Agents - ACL) [Fip02]. ACLs are interesting because they replace ad-hoc communication languages that previously were used in MAS. One important point is to distinguish in ACLs the message content and its pragmatics (i.e. metadata on the message content as well as on the intention). More specifically, the ACL message structure is composed of three levels: i) the *content level* which is often described with a content language (i.e. KIF, PROLOG, Scheme, FIPA-SL), ii) the *message level* which is defined by the performative, the ontology, the protocol, etc. used in the message, iii) the *communication level* which allows to identify the metadata on the transmission of the message itself: the sender, the receiver, and the conversation.

Communication is strongly related to agent autonomy. In order to enhance agent autonomy, it is interesting to develop communication models which expect agents to communicate without being knowledgeable of the internal believes of their partners[13], with whom they only handle output messages as an interface [Sin98].

Characteristic 6

Since the presence of a conversational process is the main difference between PD and DSG, DSG needs strong, open and dynamic communication models to manage these processes. These communication models should not limit agent heterogeneity and should allow an interpretation of messages as dynamical as possible. These models should give as much importance to acquaintances as to mental states.

In agent communication the hardest challenge is conversation modelling. Traditionally, agent conversations are modelled by communication protocols or conversation policies [Hug03, DG00]. These protocols represent the interaction structure and specify rules that must be respected during the conversation (i.e. specification of a pattern of message exchange in a conversation). Using protocols, an agent interprets messages from a conversation one-by-one, changing at each step its own state, and following the protocol to produce the next message in the conversation. The main advantage of this approach is the semantic description of the conversation (via the logical expression of preconditions, postconditions and mental states) [Gue02][14] A famous example of interac-

[12] A BDI architecture addresses how beliefs (the information an agent has about its surroundings), desires (the things that an agent would like to see achieved) and intentions (things that an agent is committed to doing) are represented, updated and processed.

[13] For example, approaches based on mental states modelling presuppose the notion of sincerity and that an agent can know what another one thinks. This is a limit to one of the primitive characteristic of agents that is autonomy.

[14] Interaction protocol is greatly bound to mental states based architecture, such as envisioned in KQML and FIPA-ACL.

tion protocol is the Contract Net protocol proposed by Smith [DS83]. Interaction protocols are either described by the language itself or are often represented with Finite State Machines or Petri Nets or Coloured Petri Nets (CPN)[15] [CCF+00]. However, this approach has weaknesses, especially interoperability, composition and verification of protocols. Agents are forced to follow a policy restricting their autonomy and the dynamic interpretation of messages. The only way for an agent to consider the entire conversation is to look at the protocol, which was previously determined (before the conversation) and which cannot change dynamically. Therefore, agents are obliged to fit fixed conversations while it should be conversations which fit dynamically changing agents.

By contrast to communication protocols, other approaches called dialogic or dialogue, for example [RPD99, JC05a], try to explain that the next answer message of a conversation cannot be foreseen, and should be determined only after the interpretation of the previous incoming message. Instead of modelling mental states and their dynamics in conversation, these approaches deal directly with speech acts rather than on hidden intentions. We further think that heterogeneous agents easily communicate not by addition of constraints on conversations but rather with an open and dynamic communication model which can fit all forms of agents and interactions. Conversational support may allow agents to handle the entire conversation dynamically and not simply interpret messages one-by-one following a previously determined structure.

Characteristic 7

DSG conversations have to be dynamic and not predetermined by a data structure that guide the conversation and limit agent autonomy (i.e. real dialogue). Communication protocols should be induced from the dialogue and not the opposite. Extraction of communication protocols is for example very important for conversations analysis and thus reasoning and learning about future conversations.

Related work on agent and Web services Some works have already been proposed using agents to enhance Web services or integrating the two approaches. [Huh02] proposes various enhancements (ontology processing, high level communication and conversation, autonomy and pro activity, etc.) to Web services through the use of agents. Huhns points out some drawbacks of Web services that significantly distinguish them from agents: they know only about themselves, and they do not possess any meta-level awareness; they are not designed to utilize or understand ontologies; and they are not capable of autonomous action, intentional communication, or deliberatively cooperative behaviour. For a precise comparison between these two concepts see also [Mor02]. According to our view on the issue, we may distinguish three approaches in agent-Web service integration:

1. **Distinct view of AAs and Web services** i.e., AAs are able both to describe their services as Web services and to search/use Web services by using mappings between agents standards and SOA standards. This approach is often based on a gateway or wrapper which transforms a standard in another one. As the main approach in agent standardisation is the FIPA one, these works always consider only FIPA agents and settle relationships between SOA and FIPA standards.

2. **Uniform view of AAs and Web services** i.e., AA and Web services are the same kind of entities. All services are Web services and they are all provided by agents (that means that the underpinning program application is an agent based system). For example [IYT04].

3. **Agent based SOA** i.e., MAS to support SOA architecture. This approach (not detailed in this paper) is not directly interested in agent service - Web service interaction but rather on the use of MAS to enhance SOAs. For example, [EMM03] discusses the use of agents for Web services selection according to the quality of matching criteria and ratings.

(1) In the distinct view of agents and Web services [LRCSN03] identifies two key concepts: i) agents should be able to publish their services as Web services for the potential use of non-agent clients; ii) these agents should advertise other entities using both a SOA standard registry (i.e. UDDI registry with WSDL entries) and an agent standard registry (i.e. FIPA DF with FIPA SD entries). This is also the case for example of [GC04], which proposes a Jade agent based on a Web Service Integration Gateway Service architecture that contains several components that operate on internal registries to maintain records of all registered services (both agent services and Web services). Greenwood's approach [GC04] takes some of the ideas generated by the Agentcities Web

[15] CPN are very interesting to model multiple parallel conversations.

Services Working Group [DHKV03] with Web Service Agent Gateway (WSAG). A drawback of this approach is that it is limited to FIPA compliant agents[16].

A particular difficult issue of this approach is communication. It consists in bridging the gap between asynchronous behaviour of agent communication and synchronous behaviour of Web services. For example, in [BV04] a Web service or an agent plays the role of a gateway that transforms a SOAP call into an ACL message. With WSAG, when a Web service invokes a SOAP call on the WSAG, the gateway transforms this synchronous call into an asynchronous FIPA-ACL message it sends to the agent that provides the service. In the same sense, the WSAG transforms a FIPA-ACL in a SOAP message but ignores the semantics, which cannot map in SOAP. Actually, this aspect is very important as the biggest risk in ACL - SOAP integration is to reduce complex and semantically richer agent communication to simple request/response of a Web service. We further think that one condition to avoid this risk, is that the SOAP - ACL transformation should be done by the agent itself and not by another entity (Web service or agent): the agent and the Web service should be the same as it is the case in the (2) approach.

(2) In the uniform view of agents and Web services, [Pet05] claims that in order to integrate agent and Web service technology, components have to be designed, which map between the different mechanisms for service description, service invocation, and service discovery, in both worlds. [Pet05, IYT04] are specifically interested in mobile agents (agents which have the ability to move from one host to another with their state); They propose respectively a "Web service engine" architecture which aims to provide bidirectional integration of both technologies and "mobile Web services" as an integration of Web services and mobile agents in order for Web services to benefit of mobility and mobile agents to benefit from the ability to compose standardised loosely coupled distributed applications.

5.2.4 Distributed knowledge: Ontology and Knowledge engineering

Ontology Ontology processing is a sub-domain of knowledge engineering which deals with knowledge representation and reasoning. An ontology is described by [Gru93, GO94] as a "formal specification of conceptualization". However, this is a too large definition e.g., mathematics is also formal specification of conceptualization. More recently, the Web community proposes a new definition: "an ontology defines the terms to describe and represent an area of knowledge" [Hel04]. Thus, we could say that ontologies deal with semantics: it is the set of terms for a particular domain, including the vocabulary, the semantic interconnections, some simple rules of inference and logic. Ontologies are composed of *concepts*, *relations* and *instances*. For example, if you want to define a car, you should say: "a car is a transportation object, with four wheels, and that you need a licence to drive it. MyCar is a car". "Car" is a concept, "is a" is a relation, and "MyCar" is an instance. Concepts are generally decomposed in several worlds: physical/real (natural concepts, physical objects), mental/imaginary (mental abstraction of the physical world), roles (behavioural descriptions), abstract (without instances), concrete/occurrence (that have a real existence). Classical relations in ontologies are for example: subclass, superclass, part of, has part, sibling, equivalence and other ones. For a good (and quite complete overview of ontologies see for example [SS04]. We may distinguish four semantic levels of ontologies:

Dictionary (definitions associated to concepts)

Taxonomy (specialization relationships (inheritance) between concepts)

Thesauri (adding to taxonomies various lexical relationships (hyperonimy, synonymy, patronimy etc.)

Ontology (adding to thesauri other kinds of axiomatic relationships (inheritance, aggregation, instantiation, ownership, causation, contains) and allowing reasoning)

An important aspect of knowledge modelling and ontologies is reasoning. Reasoning is the real use of AI in knowledge engineering. In reasoning, we could distinguish two types of approaches: i) the problem solving one, where problems are classified as well as Problem Solving Methods (PSM), methods to resolve these problems (PSMs are pre-determined and triggered when the situation requires it [FM01, Bre97, Mus98]; ii) the (machine) learning one, where logic based methods, algorithms are executed on data/information the system has and by rules (abduction, deduction, etc.) to change it. Ontologies mainly applied the first approach; they are related to PSMs.

[16] One of the force of the agent paradigm is the diversity and heterogeneity of agents. Standardization is a strong aspect for Web service interoperation but non-standardization is one of the reason for which MAS are today so different and able to address a large scope of problems. Agents and Web services need each other exactly because they come from different universes (i.e. industry for Web services and academy for MAS).

Tools for ontology development are very important, they support the information management by graphical interfaces that hide awful syntax and allow the user/designer to concentrate on the ontology itself and not on the representation formalism. Some ontology development tools are: Protégé, Triple-20, OILed, Webonto. Ontology creation and management is difficult, especially if we consider that by definition, ontologies are made to be shared and therefore used and modified by several entities. Every entity should map its own resources onto the concepts of the shared ontology. Actually, collaborative ontology development [REFtocome] is still a challenge is knowledge engineering. Introducing time in ontologies is another challenge.

Ontologies are used each time we need a semantic description: knowledge systems, information retrieval (annotating/indexing documents), knowledge reasoning, etc. We detail here two important applications of ontologies relevant for the concept of service.

The Semantic Web The recent main framework for ontologies is the Web. Today's Web is conceived for use by people (HA) who can interpret the content of the information because they have the necessary background knowledge, which they share with the creators of the given pages. Programs or AA can process this information only in ad hoc ways. As Tim Berners-Lee (one of the Web pioneers) says [BLHL01] "most of the Web content today is designed for humans to read, not for computer programs to manipulate meaningfully (...) The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation." From a simple point of view, the Semantic Web consists in adding all objects of the Web labels that represent the object. These labels are indexed in ontologies. They provide greater expressiveness when modelling domain knowledge and can be used to communicate this knowledge among agents of any types. In addition, the Semantic Web provides the necessary techniques for reasoning about ontologie concepts, as well as resolving and mapping between ontologies. In order to allow semantic based services and information management, the Web needs protocols and standards that enable specification, access, and maintenance of the meaning of terms and objects belonging to Web pages (labels). The current components of the Semantic Web framework are principally Resource Description Framework (RDF) (`www.w3.org/RDF/`), RDF Schema (RDF-S) and the Web Ontology Language (OWL) [McG04] (previously DAML+OIL).

Web service discovery, invocation, composition, interoperation is limited to be human interpretable by their lack of semantics. Existing SOA standards only provide descriptions at the syntactic level. Since no explicit semantic information is normally defined, automated comprehension of the service description is limited to cases where the provider and user assume shared knowledge about operations. Semantic Web Services (SWS) are instead semantically described Web services. They use ontologies to constitute the knowledge-level model of the information describing and supporting the use of the services. These ontologies enable automated understanding of their capabilities. More generally this semantic layer helps Web service discovery, invocation, composition, or interoperation. For example, Web service discovery should be based on the semantic match between a description of a service demand, and a description of a service offer in a semantically competent registry.

The main current approaches for SWS development are proposed by the OWL-S (OWL-based Web Service Ontology) (`www.daml.org/services/owl-s/`) and WSMO (Web Service Modeling Ontology) (`www.wsmo.org/`) working groups. OWL-S is an agent-oriented approach to semantic Web services, coming from DARPA with DAML-S and providing fundamentally an ontology for describing Web service capabilities. WSMO (inspired from the Web Service Modeling Framework [FB02]) aims at providing an appropriate conceptual model for developing and describing services and their composition, based on the principles of maximal decoupling and scalable mediation. WSMO four basic elements are ontologies, goals, services and mediators.

Semantic basis for dialogue The most important role of ontologies is as a reference for mutual understanding. A shared representation is essential to successful communication and coordination. The ontology plays the role of the "common sense" in human discourse. For example, in MAS, ontologies are used to realise the semantic level of agent communication. ACLs, such as FIPA-ACL use ontologies to support the interpretation of the content expression by the receiving agent.

Characteristic 8

In addition to syntax, DSG should handle semantics of data, information and knowledge through the use of ontologies. A semantic level should exist in dynamically constructed service descriptions (offer) and user need specifications (demand) but also in message exchanges.

5.2.5 Distributed resources: the Grid

Limits of the Web for SOA Even if DSGS should be Web oriented and compliant, we are aware that today's Web was originally proposed for document exchanges and has limits for SOA. As a first example, as previously explained, the basic Web does not deal with semantics. As a second example, the HTTP protocol (one of the main protocols of the Web) is volatile by nature: a connection is simply opened for a request/answer messages exchange. The communication is locking and synchronous. HTTP (and therefore most of Web services) is moreover stateless and thus unable to manage a long lived interaction such as conversations[17]. As a third example, Web applications adopt often the client/server approach: the most centralised mode of distributed system. In client/server mode, most of the information resides on one side (server) and most of the intelligence on the other (client). The Web is still limited to receive high level, semantically described, peer-to-peer mode architectures. For these reasons, the Web seems sufficient for storing and retrieving information, but not for real service exchange, such as it is needed in DSG. Without abandoning the Web, another infrastructure seems required. The Grid is the first distributed architecture (and infrastructure) really developed in a service oriented perspective.

The essence of the Grid concept is nicely reflected by its original metaphor: the delegation to the electricity network to offer us the service of providing us enough electric power as we need it when we need it even if we do not know where and how that power is generated. At the end of the month, I pay a bill that corresponds to my consumption. The Grid aims to enable "flexible, secure, coordinated resource sharing and coordinated problem solving in dynamic, multi-institutional virtual organisation" [FKT01]. Actually, it was originally designed to be an environment with a large number of networked computer systems where computing (Grid computing) and storage (data Grid) resources could be shared as needed and on demand. Grid provides the protocols, services and software development kits needed to enable flexible, controlled resource sharing on a large scale. This sharing is, necessarily, highly controlled, with resource providers and users defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A Grid system is naturally highly dynamic and should be able to adapt at runtime to changes in system state as resource availability may fluctuate. Such fluctuation may result from connection/disconnection of computing resources, human interaction/interruption on the computers, etc. Grid was firstly limited to computing and storage services but after extended to any kind of service insofar this is based on the dynamic allocation of virtualized resources to an instantiated service. Recently, the same semantic level add-on objective that occurred in the Web, occurred also in the Grid domain. For an introduction to the concept of Semantic Grid, see [RJS01, Gel04]. The Grid is also used as a "learning Grid" where Grid architecture and principles are used to change ITS and e-learning paradigms [ACR+05].

Characteristic 9

> *DSGS should be Grid oriented as it is the first distributed architecture (and infrastructure) really developed in a service oriented perspective providing secure and dynamic protocols for resource sharing and dynamic allocation of virtualized resource to instantiated services.*

Virtual organisation An aspect introduced by Grid systems that is fundamental for DSG is the concept of *virtual organisation*. Grid users are members of virtual organizations (also called communities). A virtual organization is a dynamic collection of individuals, institutions and resources bundled together in order to share resources and services as they share common goals. It should be viewed as a group of people who share common interests or participate to a common enterprise, and who assemble, collaborate, and communicate in a loose, distant, virtual way, using network communication facilities, tools and resources. The social perspective of DSG is very important as services define relationships within communities (cf. section 2.1.3). Grid provides the infrastructure that formalises these aspects.

Characteristic 10

> *DSGS and service users have to be organised in virtual communities. DSG creates relationships within communities of users where service exchanges take place for resolving one community member problem.*

Grid service Grid technologies have evolved from ad hoc solutions, and de facto standards based on the Globus Toolkit, to Open Grid Services Architecture (OGSA) [FKNT02] which adopts Web service standards and extend services to all kind of resources (not only computing and storage). Foster et al. call service [FKNT02]: *A*

[17] Cookies used by webmasters to remember users settings are non sufficient for conversation based service! However, this is currently evolving with HTTP 1.1 which allows sequence of request/answer and HTTPR which starts dealing with state.

(potentially transient) stateful service instance supporting reliable and secure invocation (when required), lifetime management, notification, policy management, credential management, and virtualization. OGSA introduces two major concepts in SOC: the management of state (explicit service creation) and lifetime management (the transient aspect of service). OGSA distinguishes between service factory and service instance. Whereas Web services have instances that are stateless and non-transient, Grid service instances can be either stateful or stateless, and can be either transient or non-transient.

More recently, the Web Service Resource Framework (WSRF) [FFG+04] defines uniform mechanisms for defining, inspecting, and managing stateful resources in Web/Grid services. The motivation of WSRF comes from the fact that even if today Web services successfully implement ad-hoc applications with state, they need to be standardized to enhance interoperability. WSRF replace the OGSA underlying infrastructure (OGSI) by defining standards with respect to current SOA standards. The purpose of WSRF is to define a generic and open framework for modelling and accessing stateful resources using Web/Grid services. WSRF identifies three types of service:

- A stateless service implements message exchanges with no access or use of information not contained in the input message. These are represented as pure functions easy to compose;
- A conversational service implements a series of operations such that the result of one operation depends on a prior operation and/or prepares for a subsequent operation. The behaviour of a given operation is based on processing preceding messages in the logical sequence. These are the most generic stateful services. Hard to be realized within a distributed and asynchronous context, heavy to be supported and maintained (HTTP sessions, and cookies, because HTTP does not have state);
- A stateless service that acts upon stateful resources provides access to, or manipulates a set of logical stateful resources (documents) based on messages it sends and receives.

Stateful resources are elements with state, including physical entities (e.g. databases, file system, servers) and logical constructs (e.g. business agreements, contracts) that are persistent and evolve because of service interactions. In WSRF a stateful resource is defined to: i) have a specific set of state data expressible as an XML document; ii) have a well-defined lifecycle; and iii) be known to, and acted upon, by one or more Web services.

WSRF explicitly addresses the third type. These services are modelled by an association between two entities: a stateless Web services, that do not have state, and stateful resources that do have state. WSRF calls the resulting association a WS-Resource. Both stateful resources and stateless services can be members of several WS-Resources. However, a WS-Resource is unique. WSRF is a set of six specifications that define WS-Resource[18].

Characteristic 11

DSGS should be able to manage stateful resources as state is a fundamental aspect of interaction and interaction is the basis of DSG. The state plays also the role of history. Storing the state in an external resource interacting with a stateless service seems a good principle.

Characteristic 12

DSGS should be able to manage its lifetime as it is a key aspect of dynamicity which illustrates the transient aspect of DSG.

5.3 Other domains of Informatics

5.3.1 Interaction

Several researchers consider interaction as a key element of Informatics: Turing, Milner, Hewitt, Wegner etc. We especially appreciate Wegner's approach that consists in shifting from an algorithmic view into an interactive view of Informatics [Weg97, GSW01, WG03, GW04]. Wegner's analysis returns to the original work of Turing about *choice machines*. Choice machines extended classical Turing Machine (TM) to interaction, allowing a human to intervene in the computation for doing choice that changes the rest of the computation. Unfortunately, choice

[18] i) WS-ResourceLifetime, which allows a user to specify the period during which WS-Resource definition is valid (mechanisms for WS-Resource destruction (immediately or scheduled)); ii) WS-ResourceProperties which defines how a WS-Resource can be queried and changed using Web service technologies; it allows clients to build applications that read and update data. The WS-Resource properties document acts as a view on, or projection of, the actual state of the WS-Resource; iii) WS-Notification, WS-RenewableReferences, WS-ServiceGroup, WS-BaseFaults.

machines were historically forgotten and the Turing thesis myth created: TM models all computation rather than just functions. However, as Turing himself said, Turing Machines could not provide in principle a complete model for all forms of computation i.e. computers cannot be reduced to Turing Machines [GW04]. That is why interaction needs to become more and more important in computing aiming to overpass calculus (shift from algorithms to interaction).

Wegner explain that: "It is time to recognize that today computing applications, such as web services, intelligent agents, operating systems, and graphical user interfaces, cannot be modelled by TMs; alternative models are needed" [GW04]. He explains that computation is not anymore executed in a closed world but in open system where many inputs and outputs may be interleaved one another without being all defined before computation begins. Wegner's analysis seems very important for DSG. DSGS are among those systems such that behaviour may not be modelled and represented by a TM.

Characteristic 13

DSG systems should be highly interactive as the main difference between PD and DSG is the conversational process. Interaction should be intrinsic to the underlying model of DSGS.

5.3.2 Non-determinism

A function (or an algorithm or a system) is deterministic if the result (i.e. outputs elements) is only determined by the entry (inputs elements). Any deterministic algorithm produces the same result for the same entry. During a deterministic computation for every step of the computation there is only one possibility for the following computational move.

Nondeterministic program execution and termination is not new in computer science since it has been introduced by Dijkstra in 1975 [Dij75]. One should speak of a non-deterministic system if from a certain state of the system there are several possibilities of choice for the next state. Thus, non -determinism means something not either determined or settled by previous information. Thus, a non deterministic computation permits more than one choice of the next move at some step in a computation. Besides, this calculation may have different results. Non determinism is encouraged in languages such as ADA or Constraint Style Programming (CSP) languages and algorithms. The key idea is that expressions in a nondeterministic language can have more than one possible value. With nondeterministic evaluation, an expression represents the exploration of a set of possible worlds, each determined by a set of choices. Actually, programs have different possible execution histories and should backtrack in these histories. According to Wegner, the agent limited perception of the environment guarantees a non-deterministic behaviour of MAS (open world).

Characteristic 14

DSGS should be non-deterministic. Behaviour of the system could not be predicted (thus expressed by a procedure) as it will depend on a non-deterministic conversation.

5.3.3 Machine learning and e-learning

DSG supports the creation of new knowledge; this knowledge is therefore learnt and understood by agents that generate it. Thus DSG implies learning. If we consider the two types of agents implied in DSG, we distinguish between i) machine learning and reasoning for AA; ii) e-learning, i.e., knowledge, information, know-how, or skill creation for HA. This section deals with the two as they are closely bound one another.

Concerning e-learning, we do not consider DSGS as ITS that means systems whose goal is to guide learners in their learning process thanks to a tutoring mechanisms. There are two roles in ITS, tutor and learner, and learning is always done in one sense (from tutor to learner). Actually, DSGS are rather highly interactive systems and interaction plays the key role in the learning process for both agents implied in the DSG. Interaction between two entities implies a change of state on the interacting entities. In DSG, this change of state is viewed as learning. When a service user interacts with a DSGS both of them change. This view allows the knowledge creation process to be unlimited, and to (ac)cumulate upon time: the two entities learn from each other during the interactions and re-inject what they learned into the loop.

This paragraph details a simple overview of learning, considering agent learning, i.e. reflecting about what an agent learns by itself and by/about its interlocutors. Three types of learning are distinguishable:

- The first one is *internal* (for HA) or *machine learning* (for AA). Considering AA, it is the classical approach as old as AI itself. Agents learn from a knowledge base (KB) by abstracting and generalizing, applying some abduction, deduction, induction, rules on the KB. The most widespread approach in machine learning is reinforcement learning reported in [KLM96].
- The second one is *learning-by-being-told*. It comes from DAI and is issued from the instructional effect where an agent changes or creates representations according to received messages.
- The third one is *learning as a side effect of communication*. It is quite "serendipitous"[19] as it occurs when an agent learns without being aware of it, when it learns because of a beneficed communication and collaboration context or any kind of a posteriori useful interaction with any agent not specifically called a priori for the purpose of mutual teaching or learning.

DSG considers these three types of learning. They can be all applied to any type of agent. Future DSGS should be able to adapt dynamically to service user needs. These learning capacities are fundamental for their development.

Characteristic 15

DSGS should support knowledge creation. This is both true at the agent individual level (change of state of the service user and service provider should be assimilated to learning) and at the social level (a new knowledge shared and common between agents emerge from the DSG).

Apart from this learning occurring dynamically (locally and globally) during the generation process, DSGS have also persistent states or KB they should enrich after every DSG and on which they must learn and reason. Insofar this KB keeps the history of past experiences, it may become as source of optimisation in subsequent interactions.

Characteristic 16

DSGS should be able to reason and learn on their states and history in order to evolve and improve the way they provide dynamically generated services.

5.3.4 Human integration

The integration of humans in the loop strongly depends on these domains such as Human Computer Interaction, Natural Language Processing, Dialogue processing, Visualisation, Emotion modelling etc. that improve the way humans and machines interact with each other. No matter, the model or the architecture of a system, if human users are implied, then these domains will play an important role. Their real challenge is to construct interfaces that do not limit the user expressivity and, if possible, that enhance it. Currently, pop-up means, very popular on the Web, limit the way humans may express their ideas. DSGS should process natural language (written and spoken) to be able to transform human user expressions into a formalized interpretation the system can understand and deal with. Adaptation should come from the service provider and as little as possible from the user. DSGS should also use the same way to answer users they use to request them.

Characteristic 17

DSGS should propose highly human suitable interfaces and visualisation mechanisms. Moreover, they should be able to understand and deal with the languages employed by human users. More generally, they should deal with all domain of Informatics that go toward a better integration of humans.

5.4 Dynamic concepts of programming

DSGS should be highly dynamic systems. This section presents some dynamic concepts of programming that seem to be good concrete tools to implement DSGS, assuming of course they are better integrated in higher level design and implementation methodologies.

[19] From serendipity: the faculty of making fortunate discoveries by accident.

5.4.1 Dynamically interpreted languages

Defining a computer language means to define an execution machine for this language. An execution machine takes as input a program source and returns as output the result of its evaluation. Two types of execution machines exist: either a program is compiled and then executed by a specific (eventually virtual) machine (e.g., C++, Java), or it is interpreted directly by another program, named interpreter, or evaluator (e.g. Lisp, Scheme, Prolog) [ASS96]. The two techniques have both advantages and drawbacks (addressing, syntactic analysis, binding etc.), but it is easily comprehensible that interpreted languages are more dynamic than compiled and executed languages. The compilation process ties a program content once for all. It perfectly maps with fixed and predetermined algorithms. For example, the dynamic modification of the execution machine at run time (during the evaluation of an expression) is easily feasible with interpreted languages such as Scheme [ASS96] See for example [JC05b] where STROBE agents dynamically change the message interpreter while interpreting stream of messages.

5.4.2 Streams

As [ASS96] explains, streams allows to model systems and changes on them in terms of sequences that represent the time histories of the systems being modelled. They are an alternative approach to modelling state. As a data abstraction, streams are sequences as lists are. The difference is the time at which the elements of the sequence are evaluated: list elements are evaluated when the list is constructed whereas stream elements are evaluated only when they are accessed. Streams are a clever idea that allows to use sequence manipulations without incurring the costs of manipulating sequences as lists. The basic idea consists in constructing a stream only partially, and to pass the partial construction to the program that consumes the stream. Stream processing allows to model stateful systems without ever using assignment or mutable data. It is often easier to consider the sequence of values taken on by a variable in a program as structure that can be manipulated, rather than considering the mechanisms that use, test, and change the variable. This has important implications, both theoretical and practical, because we can build models that avoid the drawbacks inherent in introducing assignment (side effects). Having stateful programs without the assignment problem should be very interesting for DSG insofar stateless services are easier to combine one another in composition and processes. The stream approach is also defended in [McC89, GSW01].

5.4.3 Lazy evaluation

For the applicative/functional language community a stream could be seen as a list which is built using a non-strict constructor. Peter J. Landin first originated the idea of a non-strict data structure. Most applicative/functional languages as Scheme [ASS96] or LISP are applicative order languages, that means that all the arguments of functions are evaluated when the function is applied. In contrast, normal order languages (such as Daisy [Joh89]) delay evaluation of function arguments until the actual argument values are needed. Delaying evaluation of function arguments until the last possible moment (e.g. until they are required by a primitive, or printed as an answer) is called lazy evaluation. A lazy evaluator only computes values when they are really required avoiding to compute values that are not really needed. For example, lazy languages allow to define the if special form as a classical function. Lazy evaluation is relevant to express the natural delayed aspect of interactions: an agent may delay the production of the next message until it interprets its partner's reaction to its previous message. [JC05a]. Thus, lazy evaluation seems a good idea for the development of the DSGS in which one cannot anticipate, or foresee messages that are going to be exchanged during conversational processes.

5.4.4 Dynamic typing

In dynamic typing (or weak typing opposed to static/strong typing) languages, neither the variable name, nor the binding between the variable name and its value contain any information of type. This information is only stored in the value itself. Dynamic typing is especially interesting for generic procedures i.e., procedures that can be applied to any type of arguments. Such procedures are interesting for DSGS because they avoid to limit service generation to well specified remote procedure call. Dynamic typing and lazy evaluation has been defended as necessary on the Web [Cer99a], where URI may be considered variables names in a global repository while XML documents represent variable values that include type information as the tree of tags.

5.4.5 Continuations

Continuation is a fundamental notion in programming languages. It is part of the execution context as well as the interpreter and the environment. When a program is interpreted, the continuation is the following move of the current process i.e., the next expression to evaluate with outputs from the current evaluation. Continuation passing style allows to dynamically manage the evaluation progress of programs and capture or change (by escape or resume) its future. In this sense continuations should be very useful for DSGS. For example, in order to solve the problem of the Web memory, [Que00] proposes to use Web continuations: instead of thinking in terms of state and transitions from page-to-page, an alternative view is proposed, where a program is suspended and resumed, continuations automatically reifying the whole state of the computation.

5.4.6 Constraint Satisfaction Programming

Constraint satisfaction programming [Dec03] consists in assigning values to variables subject to restrictions (constraints) on the set of values they can take. A solution of a constraint satisfaction problem is an assignment of all variables of the problem such that all constraints are satisfied. A constraint network is defined by a three set X, D, C that are respectively the set of variables of the problem, the set of domains for these variables, and the set of constraints on these variables, as well as a solver which executes a specific algorithms (retrospective methods: backtracks, backjumping and prospective methods: look-ahead, arc-consistence, propagation...). The three sets X, D, C come from the problem modelling phase. During DSG, this modelling phase or simply constraint expression should be done interactively with the user.

Characteristic 18

DSGS should use as much as possible techniques and methods allowing a dynamic behaviour of the system. Dynamic concepts of programming such as interpreted languages, stream processing, lazy-evaluation, dynamic typing, continuation passing style, constraint satisfaction programming are promising examples.

6 Discussion and perspectives

As we explain in the introduction, characteristics n°5 and n°9 are fundamental in this listing as they imply 80% of the other ones. Table 6 summarizes these crossed implications. From this analysis we may deduce the importance of agent approaches as well as Grid based approaches in service scenarios in order to go toward DSG. However, as some characteristics are implied only by C5 or C9 exclusively, this analysis may motivate researcher to be interested in an integration of these two approaches. [FJK04] firstly concretely explained why MAS and Grid need each other since the two approaches developed significant complementarities. They defended the concept of service as key of this integration: "A core unifying concept that underlies Grids and agent systems is that of a service: an entity that provides a capability to a client via a well-defined message exchange".

Table 2: Other characteristics implied by the agent and grid oriented characteristics

Charac	1	2	3	4	6	7	8	10	11	12	13	14	15	16	17	18
Agent (C5)		X	X	X	X	X	X	X	X	X	X	-	-	X	-	-
Grid (C9)	X							X	X	X		-	-		-	-

Our work in [REFtocome] shows and formalises how the concept of service should be the glue of this integration between MAS and Grid. It proposes a common formalisation of MAS and Grid that integrates the two approaches by defending an execution of services in a Grid context by software entities that are agents.

However, simply viewing Grid services as resource allocated interfaces of agent capabilities is for us not enough to realise DSG. The underlying agent model ought to be really interaction oriented and developed in a DSG perspective. For that reason we proposed the STROBE agent communication and representation model[Cer99b, JC05b]. In STROBE, generic communication may be described by means of *STReams* of messages to be exchanged by agents represented as *OBjects* exerting control by means of procedures (and continuations) and interpreting messages in multiple *Environments*. The model is highly influenced by functional programming constructs such as first class objects, streams, environments etc. For STROBE agents, message interpretation is done in a

given *Cognitive Environment*[20] and with a given *Cognitive Interpreter* both dedicated to the current conversation. Moreover, communication enables agents to dynamically change values in an environment and especially change (at run time) these interpreters to adapt their way of interpreting messages (meta-level learning). In the STROBE model, the aim is to put at the centre of the agent architecture the communication contexts in which it interprets messages from its interlocutors. The agent capabilities, eventually published as services, are situated in Cognitive Environments. A STROBE agent has state and capabilities dedicated to each agent it has a communication with, enabling to store, adapt, customize and personalize the way it provide and request services as it is needed for DSG.

In the STROBE and Grid integration [REFtocome], the key idea is to map the service instantiation mechanism of Grid (which allows state and lifetime management of service) with the creation of a new Cognitive Environment dedicated to another agent. The STROBE model is thus presented as a toolkit to go towards DSG. We develop this model to fit a number of DSG characteristics (n°2, 5, 6, 7, 11, 12, 13, 15, 16). For example, the experimentations (meta-level learning and dynamic specification of programs) presented in [JC05b] illustrate concrete solutions for going toward DSG. Meta-level learning enable *language enrichment* at the three levels (Data, Control, Interpreter), *dynamic specification* shows how to let the user express his own constraints concerning the functionalities of a program he needs to use.

7 Conclusion

This paper propose a reflection on the concept of service. We firstly try to understand the fundamental meaning of this concept in order to be able to develop computer-mediated contexts and systems in which human and artificial entities may realise fully the opportunities emerging from this intrinsic meaning of the concept of service. Me made a state of the art of the theoretical principles and practical technologies of services in computing nowadays and proposed a new concept called Dynamic Service Generation. The main results of the paper are the characteristics of DSG elicited in section 5. Among these characteristics, two of them where extensively described and detailed as we further think they represent the concrete computing means to realise dynamic service generation systems. They are the agent oriented and grid oriented aspect of DSGS. Services have not to be realised by simple object programs but rather by intelligent, autonomous and interactive agents able to have conversations. These agents need a secure, robust and concrete infrastructure to host the service exchanges that Grid may provide.

Besides highlighting the suitable aspect of Grid and agent for DSG, this paper aims to bring the reader to reconsider some aspect of Informatics that are really important to developed future interactive open distributed systems. The concept of service has a fundamental meaning which go very far from simply obtaining the delivery of a predetermined product. A real challenging change is needed in the way services are provided in computer-mediated contexts. What is important is not to find problems that correspond to available solutions or means but rather to deeply understand what we need and what new means and solutions will help to realise it. Actually, the DSGS development process should be itself viewed as a dynamically generated service scientists and end users exchange among themselves.

8 Acknowledgment

Section 5.2.4 about knowledge engineering and ontologies was partially inspired by a lecture given by Prof. Joost Breuker at Montpellier II University MSc and PhD students in Informatics.

Work partially supported by the European Community under the Information Society Technologies (IST) programme of the 6^{th} Framework Programme for RTD - project ELeGI, contract IST-002205. This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of data appearing therein.

References

[ACR+05] Colin Allison, Stefano A. Cerri, Pierluigi Ritrovato, Angelo Gaeta, and Matteo Gaeta. Services, semantics and standards: Elements of a learning grid infrastructure. *Applied Artificial Intelligence Journal Special issue on Learning Grid Services*, 19(9-10):861–879, November 2005.

[20] The term environment is here used with its programming language meaning, that is to say, a structure that binds variables and values.

[AHT90] James Allen, James Hendler, and Austin Tate, editors. *Readings in Planning*. Morgan Kaufmann, Palo Alto, CA, USA, 1990.

[ASS96] Harold Abelson, Geral Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, Massachusetts, USA, 2^{nd} edition, 1996.

[Aus62] John L. Austin. *How to do things with words*. Clarendon Press, Oxford, UK, 1962.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic Web. *Scientific American*, May 2001.

[Bre97] Joost Breuker. Problems in indexing problem solving methods. In D. Fensel, editor, *Workshop on Problem Solving Methods for Knowledge Based Systems, IJCAI'97*, pages 19–35, Nagayo, Japan, August 1997.

[BV04] Paul Buhler and José M. Vidal. Integrating agent services into BPEL4WS defined workflows. In *4^{th} International Workshop on Web-Oriented Software Technologies*, 2004.

[BVV03] Paul A. Buhler, José M. Vidal, and Harko Verhagen. Adaptive workflow = web services + agents. In *International Conference on Web Services, ICWS'03*, pages 131–137, Las Vegas, USA, July 2003. CSREA Press.

[CCF+00] R. Scott Cost, Ye Chen, Tim Finin, Yannis Labrou, and Yun Peng. Using colored petri nets for conversation model. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, volume 1916 of *Lecture Notes in Artificial Intelligence*, pages 178–192. Springer-Verlag, Berlin Heidelberg New York, 2000.

[Cer99a] Stefano A. Cerri. Dynamic typing and lazy evaluation as necessary requirements for Web languages. In *European Lisp User Group Meeting, ELUGM'99*, Amsterdam, Holland, 1999.

[Cer99b] Stefano A. Cerri. Shifting the focus from control to communication: the STReam OBjects Environments model of communicating agents. In Padget J.A., editor, *Collaboration between Human and Artificial Societies, Coordination and Agent-Based Distributed Computing*, volume 1624 of *Lecture Note in Artificial Intelligence*, pages 74–101. Springer-Verlag, Berlin Heidelberg New York, 1999.

[Cla05] William J. Clancey. Towards on-line services based on a holistic analysis of human activities. In P. Ritrovato, C. Allison, S.A. Cerri, T. Dimitrakos, M. Gaeta, and S. Salerno, editors, *Towards the Learning GRID: advances in Human Learning Services*, volume 127 of *Frontiers in Artificial Intelligence and Applications*, pages ??–?? IOS Press, November 2005.

[Dec03] Rina Dechter. *Constraint processing*. Morgan Kaufmann, San Francisco, 2003.

[DG00] Frank Dignum and Mark Greaves, editors. *Issues in Agent Communication*, volume 1916 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin Heidelberg New York, 2000.

[DHKV03] Jonathan Dale, Akos Hajnal, Martin Kernland, and Laszlo Zsolt Varga. Integrating web services into agentcities recommendation. Agentcities Technical Recommendation actf-rec-00006,, Agentcities Web Services Working Group, November 2003.

[Dij75] E.-W. Dijkstra. Non-determinacy and formal verification of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[DS83] Randall Davis and Reid G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.

[EMM03] Munindar P. Singh E. Michael Maximilien. Agent-based architecture for autonomic web service selection. In *1^{st} International Workshop on Web Services and Agent Based Engineering, WSABE'03*, Sydney, Australia, July 2003.

[FB02] Dieter Fensel and Christoph Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[Fer99] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison Wesley Longman, Harlow, UK, 1999.

[FFG+04] Ian Foster, Jeffrey Frey, Steve Graham, Steve Tuecke, Karl Czajkowski, Don Ferguson, Frank Leymann, Martin Nally, Igor Sedukhin, David Snelling, Tony Storey, William Vambenepe, and Sanjiva Weerawarana. Modeling stateful resources with web services. Whitepaper Ver. 1.1, Web Services Resource Framework, May 2004.

[Fip02] Foundation for Intelligent Physical Agents, FIPA ACL message structure specification, December 2002. www.fipa.org/specs/fipa00061/.

[FJK04] Ian Foster, Nicholas R. Jennings, and Carl Kesselman. Brain meets brawn: Why Grid and Agents need each other. In *3^{rd} International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS'04*, volume 1, pages 8–15, New York City, New York, USA, July 2004.

[FK99] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, USA, 1999.

[FKNT02] Ian Foster, Carl Kesselman, Jeff Nick, and Steve Tuecke. The physiology of the Grid: An Open Grid Services Architecture for distributed systems integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*. The Globus Alliance, June 2002. Extended version of Grid Services for Distributed System Integration.

[FKT01] Ian Foster, Carl Kesselman, and Steve Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Supercomputer Applications*, 15(3), 2001.

[FM01] Dieter Fensel and Enrico Motta. Structured development of problem solving methods. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):913–932, November/December 2001. See also proceedings of 11^{th} Banff Knowledge Acquisition for Knowledge-Based System Workshop, KAW'98, Banff, Canada.

[GB99] Zahia Guessoum and Jean-Pierre Briot. From active objects to autonomous agents. *IEEE Concurrency*, 7(3):68–76, 1999.

[GC04] Dominic Greenwood and Monique Calisti. Engineering web service - agent integration. In *IEEE Systems, Cybernetics and Man Conference*, The Hague, Netherlands, October 2004.

[Gel04] Marije Geldof. The semantic grid: will semantic web and grid go hand in hand? Technical report, European Commission DG Information Society Unit 'Grid technologies', June 2004.

[GO94] Tom R. Gruber and Gregory R. Olsen. An ontology for engineering mathematics. In J. Doyle, P. Torasso, and E. Sandewall, editors, *4th International Conference on Principles of Knowledge Representation and Reasoning*, Bonn, Germany, 1994. Morgan Kaufmann.

[Gru93] Tom R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

[GSW01] Dina Q. Goldin, Scott A. Smolka, and Peter Wegner. Turing machines, transition systems, and interaction. *Electronic Notes in Theoretical Computer Science*, 52(1):120–136, 2001.

[Gue02] Francis Guerin. *Specifying Agent Communication Languages*. PhD thesis, Imperial College of Science, University of London, London, UK, June 2002.

[GW04] Dina Goldin and Peter Wegner. The origins of the turing thesis myth. Technical Report CS 04-13, Brown University, Providence, Rhode Island, USA, June 2004.

[Hel04] Owl web ontology language use cases and requirements. W3c recommendation, World Wide Web Consortium, February 2004.

[Hew77] Carl Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8(3):323–364, June 1977.

[HNL02] James E. Hanson, Prabir Nandi, and David W. Levine. Conversation-enabled web services for agents and e-business. In *3rd International Conference on Internet Computing, IC'02*, pages 791–796, Las Vegas, Nevada, USA, June 2002.

[HS98] Michael N. Huhns and Munindar P. Singh, editors. *Readings in Agents*. Morgan Kaufmann, San Francisco, California, 1998.

[Hug03] Marc-Philippe Huget, editor. *Communication in Multiagent Systems, Agent Communication Languages and Conversation Poliscies*, volume 2650 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Heidelberg New York, 2003.

[Huh02] Michael N. Huhns. Agents as web services. *IEEE Internet Computing*, 6(4):93–95, 2002.

[IYT04] Fuyuki Ishikawa, Nobukazu Yoshioka, and Yasuyuki Tahara. Toward synthesis of web services and mobile agents. In *2nd International Workshop on Web Services and Agent Based Engineering, WSABE'04*, pages 48–55, New York City, NY, USA, July 2004.

[JC05a] Clement Jonquet and Stefano A. Cerri. i-dialogue: Modeling agent conversation by streams and lazy evaluation. In *International Lisp Conference, ILC'05*, pages 219–228, Stanford University, CA, USA, June 2005.

[JC05b] Clement Jonquet and Stefano A. Cerri. The strobe model: Dynamic service generation on the grid. *Applied Artificial Intelligence Journal Special issue on Learning Grid Services*, 19(9-10):967–1013, November 2005.

[Jen01] Nicolas R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.

[Joh89] Steven D. Johnson. How daisy is lazy: Suspending construction at target levels. Technical Report 286, Indiana University Computer Science Department, Bloomington, Indiana, USA, August 1989.

[KLM96] Leslie P. Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[LF97] Yannis Labrou and Tim Finin. A proposal for a new KQML specification. Technical report TR-CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland, Baltimore, Maryland, USA, February 1997. www.cs.umbc.edu/kqml/.

[LRCSN03] Margaret Lyell, Lowell Rosen, Michelle Casagni-Simkins, and David Norris. On software agents and web services: Usage and design concepts and issues. In *1st International Workshop on Web Services and Agent Based Engineering, WSABE'03*, Melbourne, Australia, July 2003.

[MBG03] Libero Maesano, Christian Bernard, and Xavier Le Galles. *Services Web avec J2EE et .NET Conception et implémentations*. Eyrolles, Paris, France, 2003.

[McC89] John McCarthy. Elephant 2000: A programming language based on speech acts. Unpublished draft, Stanford University, CA, USA, 1989.

[McG04] Owl web ontology language overview. W3c recommendation, World Wide Web Consortium, February 2004.

[Mor02] Luc Moreau. Agents for the Grid: A Comparison with Web Services (Part 1: the transport layer). In H. E. Bal, K-P. Lohr, and A. Reinefeld, editors, *Second IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID'02*, pages 220–228, Berlin, Germany, May 2002. IEEE Computer Society.

[Mus98] Mark A. Musen. Modern architectures for intelligent systems: Reusable ontologies and problem-solving methods. In C.G. Chute, editor, *American Medical Informatics Association (AMIA) 1998 Annual Symposium*, pages 46–52, Orlando, Florida, USA, 1998.

[Nif04a] Roger Nifle. Le sens du service – Analyse des sens et cohérences humaines. Le journal permanent de l'Humanisme Méthodologique, http://journal.coherences.com, July 2004.

[Nif04b] Roger Nifle. Les services sur Internet – Echec mode d'emploi. Le journal permanent de l'Humanisme Méthodologique, http://journal.coherences.com, July 2004.

[Pet05] Jan Peters. Integration of mobile agents and web services. In *1^{st} European Young Researchers Workshop on Service Oriented Computing (YR-SOC'05)*, pages 53–58, Leicester, UK, April 2005. Software Technology Research Laboratory, De Montfort University.

[Que00] Christian Queinnec. The influence of browsers on evaluators or, continuations to program Web servers. In *5^{th} ACM SIGPLAN International Conference on Functional Programming, ICFP'00*, number 9, pages 23–33, Montreal, Canada, September 2000. Volume 35, Issue 9 of ACM SIGPLAN Notices.

[RG91] A.S. Rao and M.P. Georgeff. Modelling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *3^{rd} International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484, San Mateo, California, USA, April 1991. Morgan Kaufmann.

[RJS01] David De Roure, Nicholas Jennings, and Nigel Shadbolt. Research agenda for the semantic Grid: A future e-science infrastructure. Technical report, University of Southampton, UK, June 2001. Report commissioned for EPSRC/DTI Core e-Science Programme.

[RPD99] Pierre-Michel Ricordel, Sylvie Pesty, and Yves Demazeau. About conversations between multiple agents. In *1^{st} International Workshop of Central and Eastern Europe on Multi-agent Systems, CEEMAS'99*, pages 203–210, St. Petersburg, Russia, June 1999.

[Sea69] John Searle. *Speech acts: An essay in the philosophy of language*. Cambridge University Press, Cambridge, UK, 1969.

[SH05] Munindar P. Singh and Michael N. Huhns. *Service-Oriented Computing Semantics, Processes, Agents*. John Wiley & Sons, Ltd, 2005.

[Sin98] Munindar P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, 1998.

[SS04] Steffen Staab and Rudi Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.

[Weg97] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, May 1997.

[WG03] Peter Wegner and Dina Goldin. Computation beyond turing machines. *Communications of the ACM*, 46:100–102, April 2003.

[Woo02] Michael Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, Chichester, UK, February 2002.

A SOA technologies and consortiums

Table 3: SOA technologies and consortiums URLs

CONSORPTIUMS AND TECHNOLOGIES	URL
W3C (World Wide Web Consortium)	www.w3.org/
WSDL (Web Services Description Language) SOAP (Simple Object Access Protocol) WSCL (Web Services Conversation Language) WSCI (Web Service Choreography Interface) WS-CDL (Web Services Choreography Description Language) OWL (Web Ontology Language) RDF (Resource Description Framework) XML (eXtensible Markup Language)	
OASIS (Organization for the Advancement of Structured Information Standards)	www.oasis-open.org
UDDI (Universal Description, Discovery and Integration) WS-BPEL (Web Services Business Process Execution Language) BTP (Business Transaction Protocol) ebXML registries SAML (Security Services) WSRM (Web Services Reliable Messaging)	
WfMC (Workflow Management Coalition)	www.wfmc.org/
XPDL (XML Processing Description Language)	
UN/CEFACT (United Nations Centre for Trade Facilitation and Electronic Business)	www.unece.org/cefact/
ebXML Business Process Specification Schema (BPSS)	
BPMI (Business Process Management Initiative)	www.bpmi.org
BPML (Business Process Modeling Language) BPQL (Business Process Query Language)	
Industrials (IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems etc.)	
XLANG BPEL4WS (Business Process Execution Language for Web Services) WS-Coordination (composed of WS-AtomicTransaction, WS-BusinessActivity) WS-Transaction, WS-Inspection, WS-Addressing, WS-Security WS-Trust, WS-reliability	