SOA & *PLASTIC* conceptual model

Marco Autili
University of L'Aquila

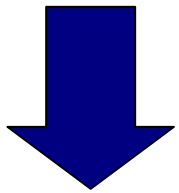
marco.autili@di.univaq.it
http://www.di.univaq.it/marco.autili

Road Map

- *What are service? [1]*
- *What are software services? [1]*
- *Service Oriented Architecture (SOA) [7]*
- *SOA characteristics [4]*
- *Component orientation VS Service orientation [2]*
- *Component orientation + Service orientation [2]*
- *State of the art on service oriented conceptual models and research direction [3,5,6,7]*
- Getting start with a service conceptual model for PLASTIC *[8]*

Services

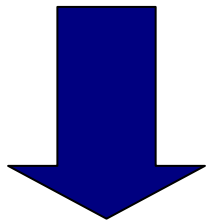
- What are *services* ?
  - One of the abstract and context independent definition is:
    - Services are labors that, if accomplished, produce a non tangible useful benefit (Software Technologies Unit of the European Commission).
- What are *software services* ?
  - Software services are functionalities provided by software applications that supply:
    - computational resources on request;
    - informational resources on request.
  - Functionalities are contractually defined in the *service description* (Software Technologies Unit of the European Commission).



- **S**ervice **O**riented **A**rchitecture (SOA)

SA, CBA and SOA …

- *Software Architecture (SA);*
- *Component Based Architecture (CBA);*
- *Service Oriented Architecture (SOA):*
  - *service designer and developer?*



  - *… concept and principles of SOA should be taken into account.*

The most important aspects of SOA

Service Implementation and Service Provider

Service Contractual Description

HOW

WHAT

Do not worry about HOW ...

... only expect that it will

Why SOA? (1/2)

- SA have attempted to deal with **increasing** levels of software **complexity** …
- … traditional architectures seem to be **reaching the limit** of their ability to deal with such a complexity.

- At the same time, traditional needs of Information Technology (IT) organizations persist:
  - The **need to respond quickly to new requirements** of business;
  - the need to continually **reduce the cost of IT** to the business;
  - the **ability to absorb and integrate** new business partners and new customer sets, ecc.

Why SOA? (2/2)

- We need an architectural framework which allows the **assembly of services** for the **rapid**, and even **dynamic**, **delivery of solutions**.
- SOA is being promoted in the industry as **the next evolutionary step in SA** to help IT organizations meet their evermore complex set of challenges.
- SOA "**could be a 2010 phenomenon**"
  - From: Financial Times, 4 May 2005, Richard Waters: "Plugging together software may soon painless"

Some motivations

- Corporate management always pushes for:
  - **better IT utilization** to elaborate and transmit data;
  - **greater Return on Investment (ROI)**:
    - i.e., a financial measure of the relative value of an investment, usually expressed as a percentage, calculated by dividing earnings produced by the amount invested).

- Systems must be developed by taking into account the complexity introduced by **heterogeneity**, **dynamicity**, **integration**, **reuse**, **adaptability**, **ubiquitous access to the heterogeneous infrastructures**:
  - **legacy systems must be reused** rather than replaced, because with even more constrained budgets, **replacement is cost-prohibitive**;
  - **new business models**, so entire IT organizations, applications, and infrastructures **must be integrated and absorbed**;
  - applications need to be **adaptive** to the context of use;
  - ...

SOA is not new!

- It is an alternative *loosely-coupled* model to the more traditionally *tightly-coupled* object-oriented models that have emerged in the past decades.
- The overall design is service-oriented but do not exclude the fact that individual services can themselves be built with object-oriented designs.
- SOA is *object-based*, but it is not, as a whole, *object-oriented*.
- The difference lies in the **"interfaces"** themselves. A classic example of a proto-SOA system that has been around for a while is the (CORBA), which defines similar concepts to SOA

SOA is not new!

- … SOA is different in that it relies on a more recent advance based upon XML.
- By describing interfaces in an XML-based language called WSDL, services have moved to a more dynamic and flexible interface system than the older IDL found in CORBA.
- Web services aren't the only way to implement SOA.
- CORBA, as just explained earlier is one other way and so are Message-Oriented Middleware systems such as the IBM MQ Series.
- .. to become an architecture model, you need more than just a service description:
  - you need to define how the overall application performs its workflow between services.

SOA characteristics: modularity

- **Modular Decomposability:**
  - break of an application into many smaller services.
  - each module is responsible for a single, **distinct functionality**.
  - Within a "**top-down design approach**", the goal is to obtain the smallest unit of service that can be reused in different contexts.

SOA characteristics: modularity

- **Modular Composability:**
  - software services may be freely combined as a whole with other services to produce new **composed services.**
  - each service is still responsible for a single, **distinct functionality**.
  - Within a "**bottom-up design approach**", the goal is to create services sufficiently independent to be reused in entirely different applications from the ones for which they were originally intended.

SOA characteristics: modularity

- **Modular Understandability:**
  - A service should be described in such a way that a consumer/developer is able **to understand the functionality and other specification** of the service without having any knowledge of other services.
  - If the **functionality provided** from the service is not understandable, the **consumer** deciding whether to use the service will have a difficult time making a decision.
  - If the **behavioral specification** of a service is tied to other specification, **developer** will have hard time to reuse the service.

SOA characteristics: modularity

- **Modular Continuity:**
  - the impact of **changes in one service should not require changes** in other services or in the consumers of the service.
  - An service description that **does not sufficiently hide the implementation details** of the service creates a domino effect when changes are needed:
    - other service may have assumption on this implementation details.

SOA characteristics

- **Interoperability:**
  - SOA stresses **interoperability**:
    - the ability of systems using different platforms and languages to communicate with each other.
  - Protocol and a data format should be standardized:
    - For instance JAX-RPC and JAXM map Java data types to SOAP.

SOA characteristics

- **Loose Coupling:**
  - The **binding** from the service requester to the service provider **should loosely couple the service**.
  - This means that the service **requester has no knowledge of the technical details** of the provider's implementation, such as the programming language, deployment platform.
  - Implementation on each side of the conversation can change without impacting the other, provided that **the message schema stays the same**.
  - **Legacy code**, such as COBOL, can be replaced with new **Java** code without having any impact on the service requester.

First service classification

- **Context adaptive and intelligent user services**:
  - These are “ambient intelligent” type of services providing users the ability to access the services they need, anywhere from any device.
- **Information services**:
  - services that provide (personalized) information, for instance by comparing, classifying, or otherwise adding value to separate information sources.
- **Intermediary services**:
  - services that help in finding appropriate services, for instance search services.
- **Location-based services (LBS)**:
  - family of service that depend on the knowledge of the geographic location of mobile stations.

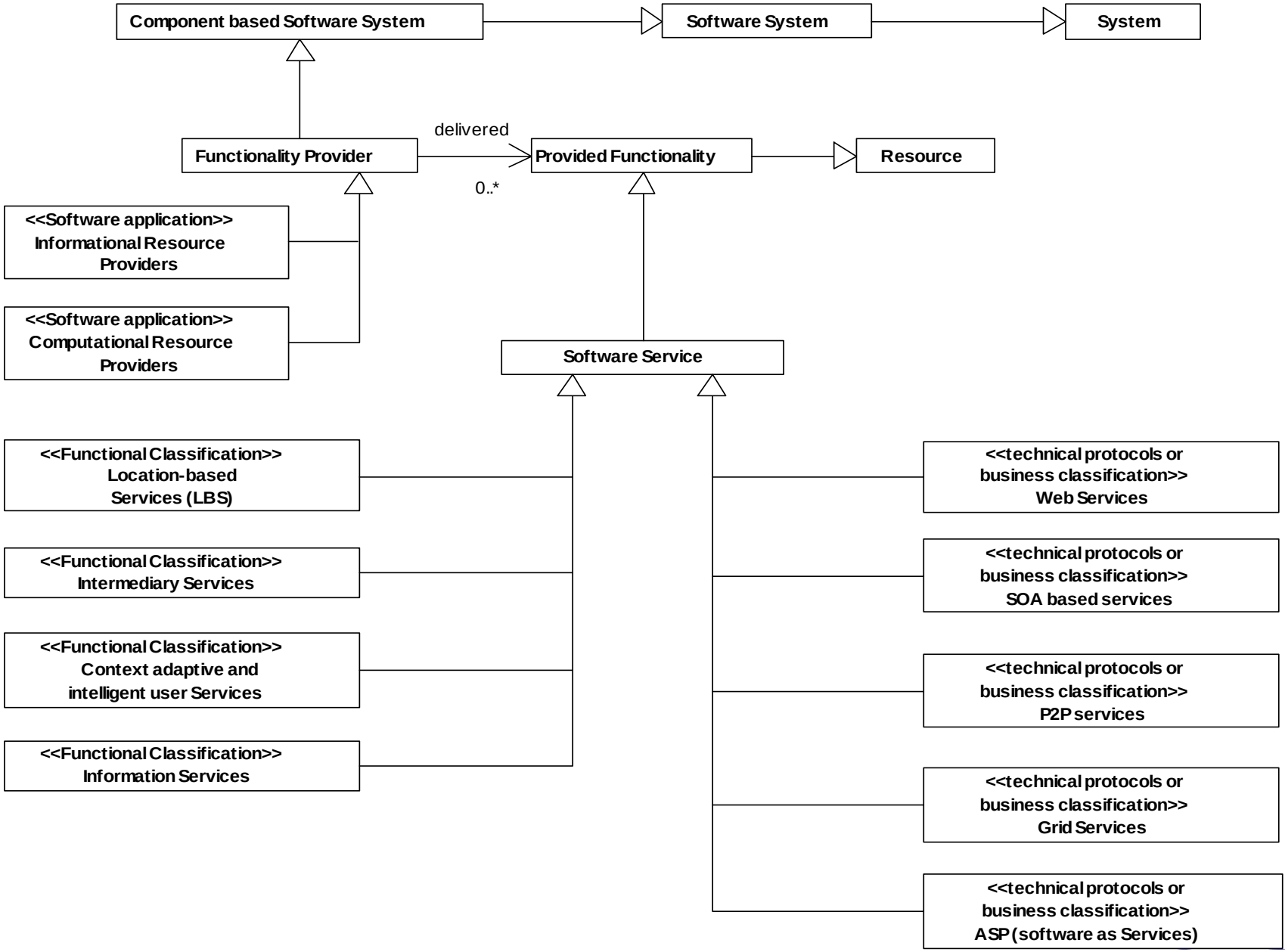
Second service classification (1/2)

- **Web Services**:
  - support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (WSDL).
  - Interact by WS description using SOAP-message (conveyed using HTTP with an XML serialization and other Web-related standards.
- **P2P services**:
  - are services “p2p” networks, meaning sharing files and content between computers via direct interaction between users on the network, facilitated by a virtual name space (VNS).
  - A VNS associates user-created names with the “physical” IP.
  - User do not need to know the location of other users.

Second service classification (2/2)

- **Grid services**:
  - Grid is a type of parallel and distributed computation based on aggregation of geographically distributed "autonomous" resources dynamically at runtime. Grid exploits ability to share and aggregate distributed computational capabilities and deliver them as service.
- **ASP, software as a service**:
  - ASP is an abbreviation of Application Service Provider, a third-party entity that manages and distributes software-based services to customers across the network.
  - In essence, ASP's are a way for companies to outsource some their need of information technology.

Service classification



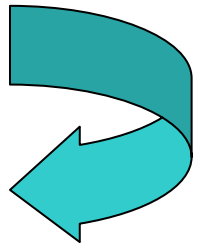
Component Orientation (1/6)

- *CBSE* promotes the *construction* of software application as *composition* of reusable building *"blocks"* called *components*.

- *"A software component is a binary unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties" (Szyperski).*

Component Orientation (2/6)

- Component Orientation makes a distinction between *developers* and *assemblers* *(third party)*.



- *Binary Unit* delivering.

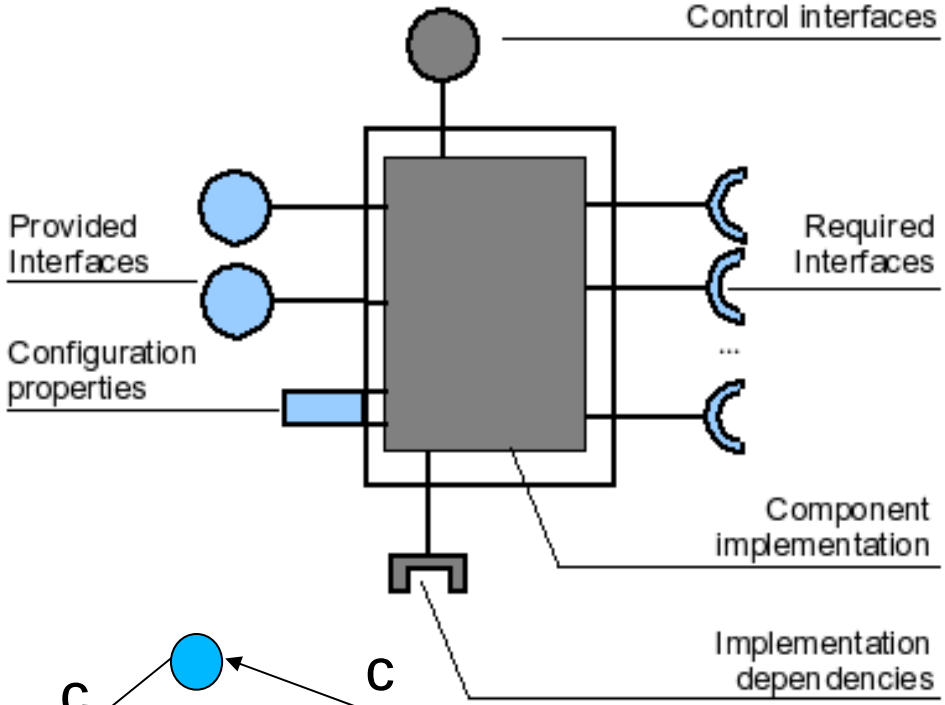
- Application assembly is based on components *physically available* to the assemblers when composing them.
  - Integration at *assembling time*:
    - deployment, instantiation and connection.

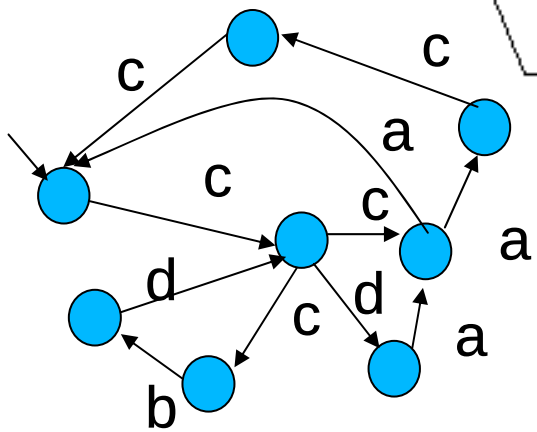
Component Orientation (3/6)

- In the majority industrial component model (COM/DCOM, EJB), component are *similar to classes*.
- Third parties instantiate component and then *connect instances* in some appropriate fashion.
- Component are often *hardly reused* as they are ….
- … *Component Adaptation* promotes the use of adaptor.

Component Orientation (4/6)

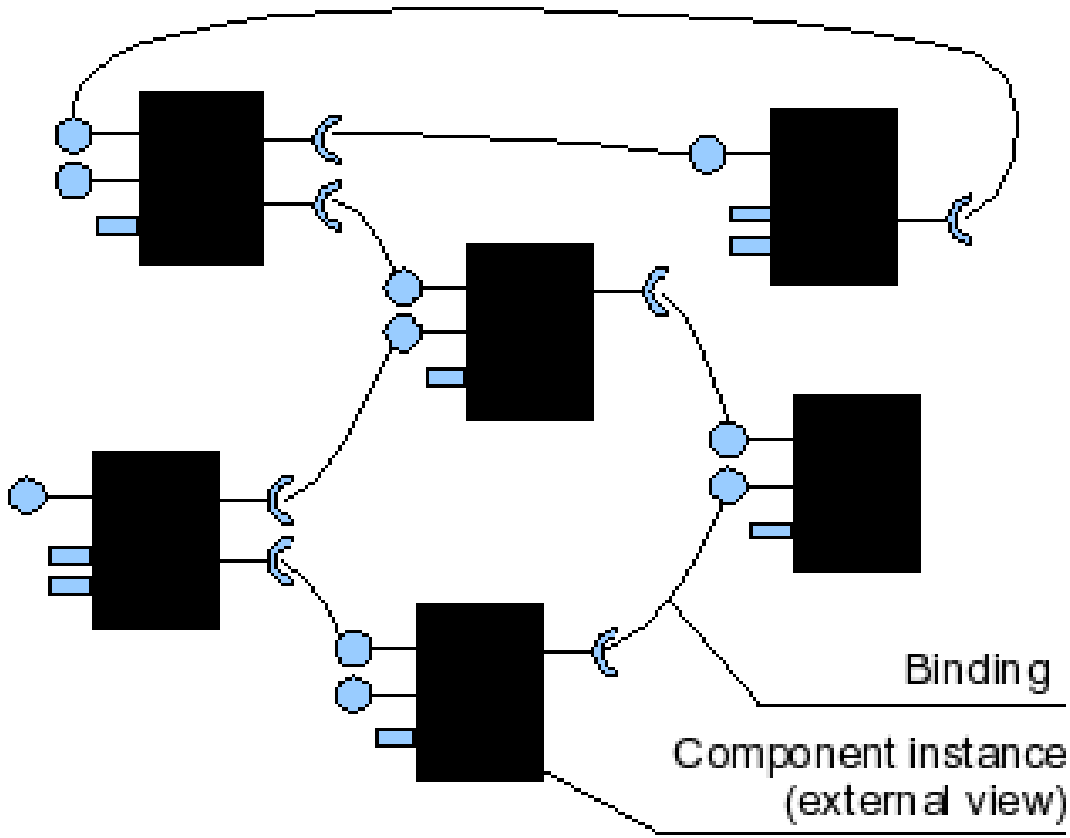
- To support *structural composition*:
  - Component instance external view;
  - behavioral specification;
  - standard programming languages or specialized ones.





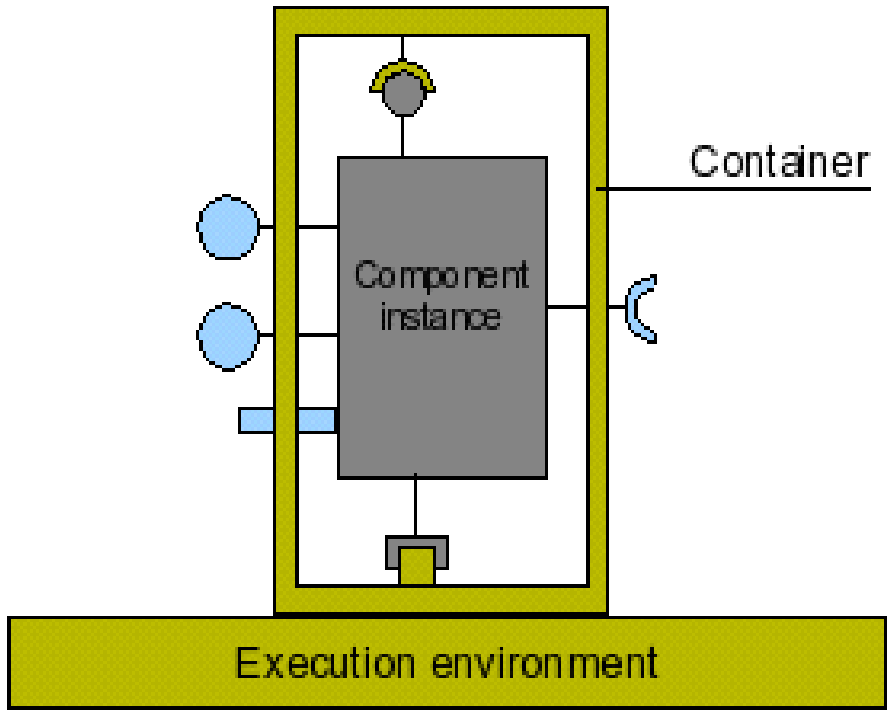
Component Orientation (5/6)

- *Hierarchical composition* is achieved when the *external view* of a component is itself implemented by a composition.



Component Orientation (6/6)

- Delivered as binary code in a package along with their required resources.
- Deployment dependencies need to be fulfilled.
- Require an execution environment (e.g., a container)

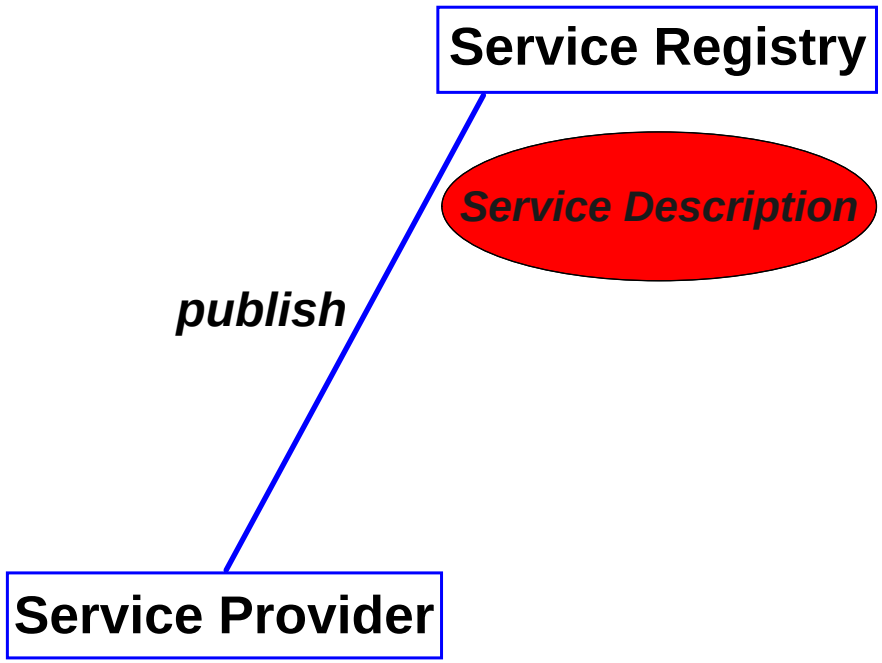


Service Orientation (1/5)

- Applications are *assembled* from reusable building *"blocks"*, but in this case the blocks are *services*.
- Services are *functionalities* defined by contractual descriptions:
  - Services functionality as mix of syntax, semantics and behavior.
- Application assembly is *based only* on *service description.*
- *The most interesting aspects are:*
  - the actual service provider is *located later*:
  - integration *prior* or *during execution*.
  - the focus is on how the service are described in order to support *dynamic discovery*.

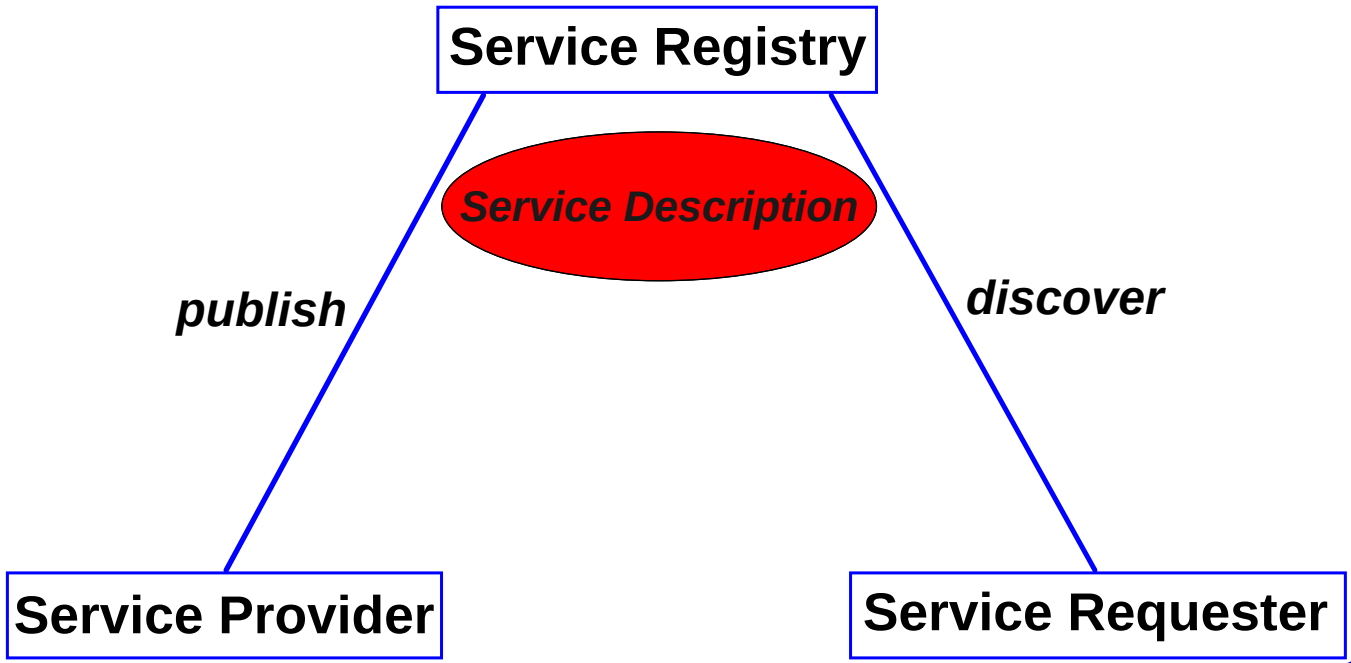
Service Orientation (2/5)

- Support of dynamic discovery (*find-bind-execute* paradigm):
  - Service Providers *publish* their *service description* into a service registry …



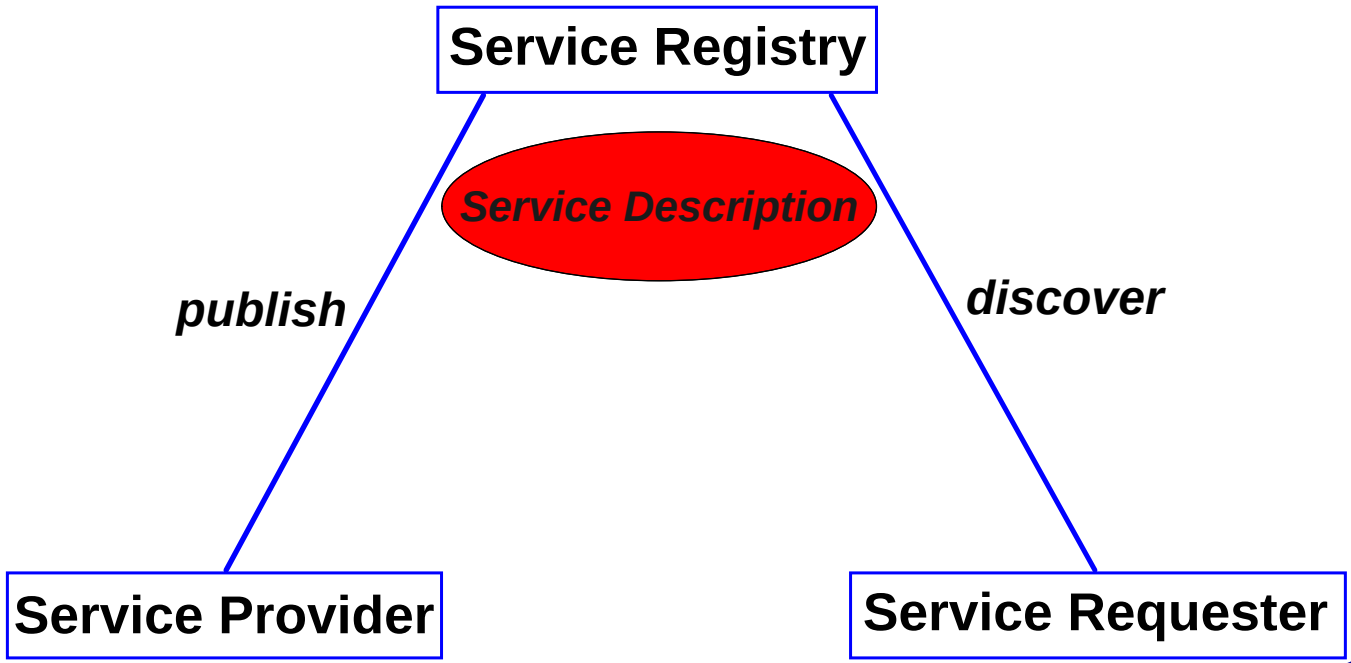
Service Orientation (2/5)

- Support of dynamic discovery (*find-bind-execute* paradigm):
  - Service Requestors interrogate the registry for a particular service description to *discover* service providers ...



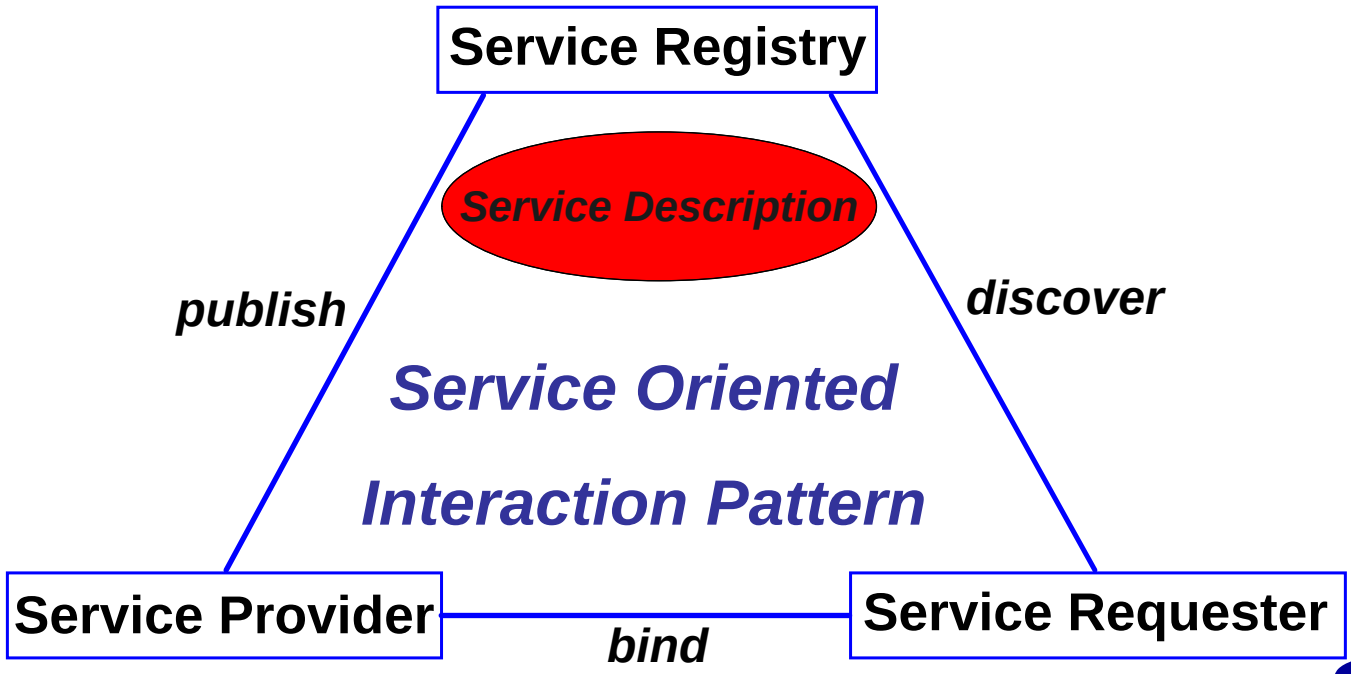
Service Orientation (2/5)

- Support of dynamic discovery (*find-bind-execute* paradigm):
  - … one ore more service provider references (if there exist) are returned …



Service Orientation (2/5)

- Support of dynamic discovery (*find-bind-execute* paradigm):
  - … after ***binding*** a service provider, a reference to the *"service object"* implementing the service functionalities is returned.



Service Orientation (3/5)

- Service requestor *is not tied* to a particular service provider:
  - service *providers may be substituted* whenever they *obey to the contract* imposed by the service description.

- Due to *dynamic availability*, a service may be removed from the registry at any time:
  - running application must be able to *releasing* (*incorporating*) *departing* (*arriving*) services.

Service Orientation (3/5)

- To deal with *dynamic availability:*
  - some service platforms provide *notification mechanisms* used to inform service requesters of the arrival or departure of the services.
  - others use the concept of a *service lease*, which means that service availability is guaranteed only for a determined time period after which the lease must be renewed.

Service Orientation (4/5)

- *Service composition* is an abstract composition:
  - it is based *only on service descriptions* and it *concretizes* only at run-time.

- *Hierarchical service composition* is achieved when a service composition itself has a service description.

Service Orientation (5/5)

- Development platforms such as the one of PLASTIC should respect the *service orientation principles*.
- Examples of other platforms include the CORBATrader, Jini, OSGi, IBM WebSphere, ecc.

Service Orientation *VS* Component Orientation

- Emphasis on software service descriptions;
- Natural separation from description and implementation;
- Only service description is available during assembly;
- Integration is made prior or during execution;
- Better support for run-time discovery and substitution;
- Native dynamic availability;
- Abstract composition.

- Emphasis on component external-view;
- No evident separation between component external-view and implementation;
- Component physically available during assembly;
- Integration at assembly-time;
- No native support for run-time discovery and difficult substitution.
- No native dynamic availability;
- Structural composition.

Service + Component Orientation

- A service-oriented component model should introduces concepts of service orientation into a component model.
- This combination facilitates *dynamism* in a component model:
  1. Service as a **contract.**
  2. Components **contractually implement** service descriptions;
  3. A service-oriented **Interaction pattern** is used to contractually resolve service dependencies;
  4. Abstract composition **in terms of contracts**;
  5. Contracts are the basis for **substitutability**;

(1) Service as a contract

- *A service is provided functionality: it is a reusable operation or set of operation.*

- *A contract defines a service's characteristics for composition, interaction and discovery. This is achieved by describing some combination of the service's syntax, behavior, semantics and QoS.*

- *An important syntactical aspect of a contract is that it explicitly declares service's "contractual dependencies" on other services to enable structural composition.*

(2) Components contractually implement service descriptions

- *By implementing a contract the component provides the described service and obeys its constraints.*
- *In addition to the service "contractual dependencies" specified in the contract, a component may also declare additional "implementation-specific" service dependencies to enables structural composition.*
- *Services are the sole means of interaction among component instances.*

(3) Interaction pattern to resolve service dependencies

- Service provided by component instances are **published into a service registry**.

- The **service registry** is used to dynamically discover services at run time for resolving service's *"contractual dependencies"*.

(4) Abstract composition

- An **abstract composition is a set of contracts** that are used to select concrete component to instantiate.
- **Explicit bindings are not necessary**, since they are inferred at run time from the service “contractual dependencies” declared in the constituent contract.
- **Composition become concrete at run time** as component that provide the constituent contracts are discovered and instantiated into the composition.
- **Composition is a continuous run-time activity** that responds to the availability of services.

(5) Contracts for substitutability

- It is **not necessary to select particular components** for a composition, since compositions are defined in terms of contract, which are basis for substitutability.
- In a composition, any component that implements a given contract **can be substituted** with any alternative component that implements the same contract.

Dynamic availability and substitutability

- Differently from both traditional component and service orientation there are two levels of dependencies:
  - **contractual**;
  - **implementation specific.**
- In component orientation, **only implementation dependencies exist**, which hinders substitutability.
- In service orientation, **contractual dependencies are not declares as part of the contract**, which eliminates the possibility of structural composition.
- **Dynamic availability** is also enabled because explicit bindings are not necessary, since they are inferred from the contracts.
- Finally, all **"service dependencies" are resolved at run time** using the service-orientation interaction pattern and this interaction pattern forms the basis of a continual composition life cycle.

Challenges: Ambiguity and Dynamic Availability

For instance:

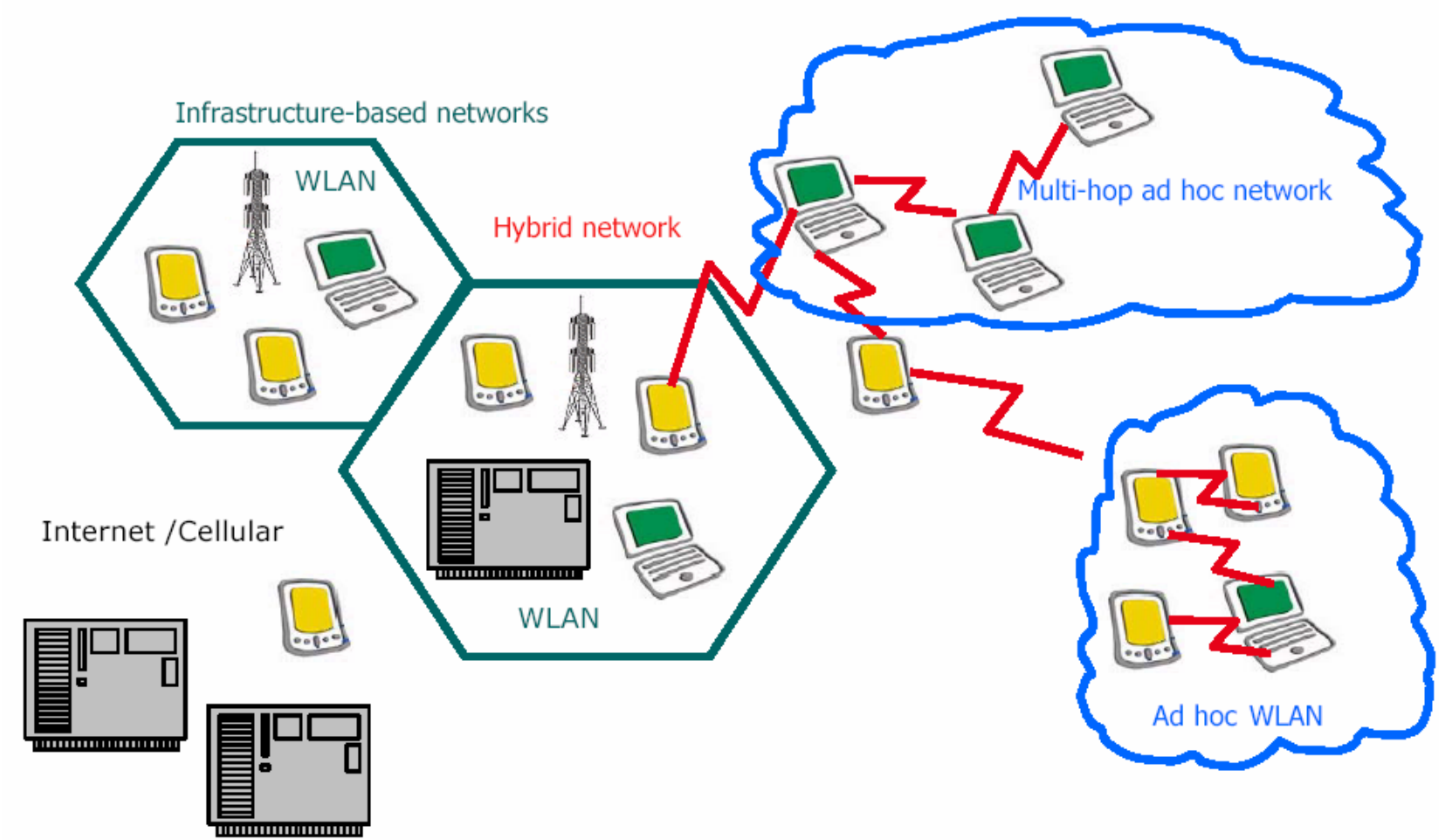
- *WSDL are primarily concerned with the description of functionality and interface:*
  - *semantics are not defined.*

- *Adaptive Service Grid (ASG) semantically rich description by functional and non-functional requirements:*
  - *Service description by the ASG language are then translated into a deployable EJB component.*

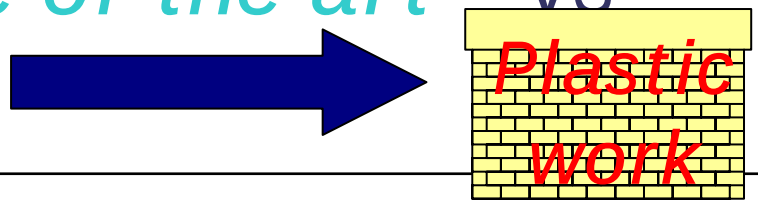
PLASTIC Vision

- Development of a *provisioning platform* for software services deployed platform for software services deployed over B3G networks.
- Basis upon both *Web services* and *standard component-based technologies*.
- *Service-oriented* and *component-based* service development.
- Applications defined as the (possible) *composition of networked services*.
- Service developed as a *composition of components*.

The B3G open wireless environment



- *State of the art* VS *PLASTIC vision*



- How should we conceptualize:
- *service.*
- *service description.* How should we describe services in order to have real world abstract services correctly represented in a description?
- *QoS?* Which QoS measurements should we take into account in order to deliver services that gain the users confidence that the services will operate as they expect?
- *monitors.* How should we monitor services in order to be sure that a service provider really supply the service as described and with the declared QoS?

Moreover

- How should we *preserve* the user expected Quality of Service when moving across different infrastructures?
- How should we tackle run time discovery and services composition to have efficient *dynamic adaptation.*
- How should we address *service adaptation to the context of use* spanning from the Provider context of use to the Consumer context of use?

SeCSE Service Oriented Conceptual Models

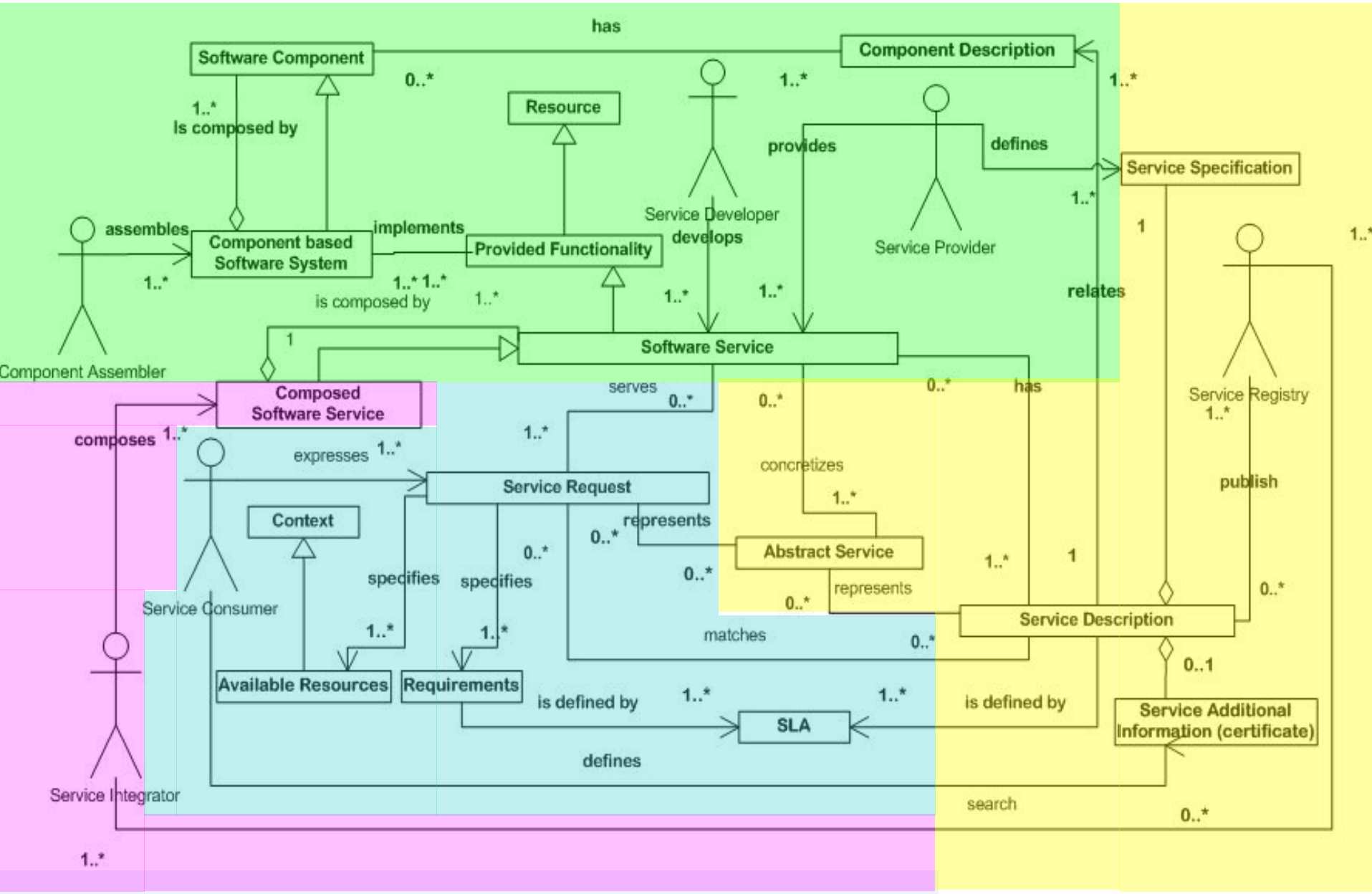
- The project *SeCSE*:
  - The "*SeCSE* **conceptual model**" (described using UML) aims at providing a common terminology across the project.
- The model has been designed to be *extensible*.
- Inspired to the model of *Web Service Architecture* (*WSA*), drafted by *W3C*, it is *complementary* since it refines part of WSA.

WSA and SeCSE **conceptual model**

- *WSA* brings into focus four aspects:
  - service (*Service Model*);
  - messages (*Message Oriented Model*);
  - resources (*Resource Oriented Model*);
  - resources and behaviors constraints policies (*Policy Model*).

- The *SeCSE* model attempts to clarify and detail the Service Model focusing on:
  - service publication, discovery, composition, and monitoring;
  - service description and semantics;
  - Agents (entity of the human real-world):
    - persons, organizations, SW systems, Services, …
  - actors (roles):
    - operation and service providers, developers, assemblers, monitors, consumer

Scratching a core conceptual model



Epilogo

- This slides are a selection of various information concerning SOA retrieved from the Web, Literature and from my own experience (see references in the next slide).

References

- **[1]** C. M. Anne-Marie Sassen. The service engineering area. *An overview of its current state and a vision of its future*, July 2005. *ftp://ftp.cordis.lu/pub/ist/docs/directorate d/st-ds/sota v1-0.pdf*.
- **[2]** Humberto Cervantes and Richard S. Hall. *Autonomous adaptation to dynamic availability using a service-oriented component model*, ICSE 2004.
- **[3]** W3C Group. Web services architecture (WSA), February 2004.
- **[4]** SUN. Service Oriented Architecture. *http://www.theserverside.com*, 2003.
- **[5]** *ASG, IST FP6 project nr 004617.*
- **[6]** SeCSE project: Service Centric System Engineering.
- **[7]** IBM, http://www-128.ibm.com/developerworks/webservices/newto/
- **[8]** PLASTIC project, Providing Lightweight and Adaptable Service technology for pervasive Information and Communication.