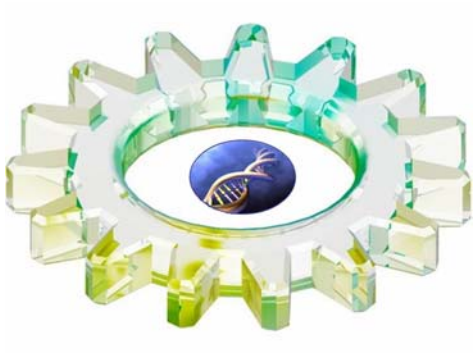

BIONETS

WP 3.2 – AUTONOMIC SERVICE LIFE-CYCLE AND SERVICE ECOSYSTEMS

TUB, UBASEL, CN, INRIA, SUN, NOKIA, VTT, TI

Specification of Service Life-Cycle

Deliverable 3.2.1

Reference:	BIONETS/TUB/WP3.2/v1.0
Category:	Report
Authors:	Heiko Pfeffer, David Linner, Ilja Radusch, Stephan Steglich (TUB), Lidia Yamamoto (UBASEL), Janne Lahti, Jyrki Huusko (VTT), Daniele Miorandi (CN), Antonietta Mannella (TI), Françoise Baude, Viet Dung Doan, Ludovic Henrio, Paul Naoumenko (INRIA), David Villegas, Juanjo Aparicio (SUN)
Editors:	Heiko Pfeffer, David Linner, Stephan Steglich, Ilja Radusch (TUB)
Verification:	Daniel Schreckling (UNIHH)
Date:	8th February 2007
Status:	Final
Availability:	Public

Executive Summary

The objective of this deliverable is to introduce a biologically inspired service lifecycle enabling services to cope with challenges arising due to the assumptions made for the future bio-inspired computing environments such as high mobility and heterogeneity with regard to devices and services themselves. The BIONETS service life therefore builds upon providing a high level of autonomy in terms of service evolution and adaptation as well as service self-control and self-management. Since services should become the main entity within a BIONETS computing environment, service interaction models are outlined able to meet the requirements implied by an autonomous service design.

Based on these objectives, a service architecture as well as a service interaction framework are modelled implementing the introduced services and service interaction schemes, respectively.

However, this deliverable does not lay claim to cover all introduced aspects entirely; it rather is intended to provide a solid and coherent starting point for more detailed investigations within different aspects of the service life cycle. For instance, deliverable D3.2.2 focuses on service evolution and deprecation concepts while deliverable D3.3.2 is supposed to contribute to aspects of service self-management and self-control.

The deliverable is thus structures as follows. After a short introduction, section 2 addresses implications for the service architecture derived from precedent deliverables. By introducing the BIONETS service life cycle, section 3 covers the core issue of this deliverable; the according interaction models are subsequently introduced in section 4. The BIONETS service life cycle is supposed to cover all aspects of autonomy of services. However, some mechanisms are not assignable to special reference points within the service life cycle, but appear to address it entirely instead. These aspects are related to the service self-management issues such a resource management and outlined in section 5. The document is finally concluded in section 6, also providing an outlook especially focusing on the roadmap outlined so far and other deliverables referring to this document.

Document History

Version History

Version	*Status*	*Date*	*Author(s)*
0.1	*First Draft*	*26.10.2006*	*TUB*
0.2	*First Contributions*	*17.11.2006*	*TUB*
0.3	*2nd Milestone Draft*	*04.12.2006*	*TUB, UBASEL, VTT, CN*
0.4	*3rd Milestone Draft*	*22.12.2006*	*all*
0.5	*Intermediate Draft*	*27.12.2006*	*CN*
0.6	*1st Review Draft*	*02.01.2007*	*TUB*
0.7	*2nd Review Draft*	*08.01.2007*	*TUB*
0.8	*Final Review Draft*	*10.01.2007*	*TUB*
0.9	*Revised Version*	*25.01.2007*	*TUB, VTT, CN, INRIA, TI, SUN*
1.0	*Final Version*	*08.02.2007*	*TUB*

Summary of Changes

Version	*Section(s)*	*Synopsis of Change*
0.1	*all*	*ToC proposal*
0.2	*2,3*	*Framework Implications and Life-Cycle Draft*
0.3	*2,3,4,5*	*First contributions of contents*
0.4	*2,3,4,5*	*Refined contributions of contents for all sections*
0.5	*3*	*Improvements by CN*
0.6	*2,3,4*	*Editing work for the review version*
0.7	*1,5,6*	*Editing work, Introduction, Outlook and Conclusion added, Service Self-Management improved*
0.8	*all*	*Final editing work for the review version*
0.9	*all*	*First draft of the reviewed and revised document*
1.0	*all*	*Final editing*

Contents

1. Introduction

1.1 Motivation and Objectives

A major goal of the BIONETS project is to investigate how large-scale Service-oriented Computing (SoC) can be realized in a decentralized manner, a direct result of the assumptions about the physical environment as introduced in [82]. The BIONETS computing environment is supposed to be composed of numerous small and mostly mobile devices forming networks in an ad hoc manner. Consequently, the services are provided and executed by the device, either alone or in a cooperative fashion, while the distribution of services is supposed to be performed directly among the devices.

Apart from the special characteristics of the computing environment, a further objective of the development of the service framework is to research how to add a significant value for users. We argue that this objective can be achieved in making the service environment self-aware and therefore adaptive. Adaptiveness is supposed to help on the one hand to make the services robust with regard to changes in the setting of the computing environment. On the other hand it enables the service environment to improve self-reliantly the functionality offered to the user. Hence, adaptiveness helps for instance to reduce the error rate, safe resources and address users' needs in a better way. Moreover, as we address a human-centric environment, we see the need to make services accessible in an easy and comfortable fashion. This includes supporting the user in accomplishing complex task through decomposing problems and utilizing all resources provided by the service environment to solve them in cooperative manner.

We investigate the applicability of biologically inspired concepts to address the challenges raised above. Consequently, this document outlines the current state of our work on bio-inspired services. Based on service architecture principles researched in WP3.1, we describe a life-cycle for services in distributed environments, which incorporates the notion of evolution. As the life cycle has a significant impact on the structure of the service environment, we additionally regard the required extensions to the service framework. This document is intended as base for the specification of the BIONETS service environment and as starting point for future improvement. Several sections already discuss multiple alternatives in approaching the solution of a certain problem. In general, concepts of D2.2.2 [80] are picked up and mapped to a SoC-conform view. Detailed consideration about service evolution, evaluation and deprecation are part of the next deliverable in this work package.

1.2 Document Outline

The structure of the document addresses two focal aspects, 1. the Evolutionary Service Life-Cycle, 2. the corresponding extensions/refinements of the service framework.

Section 2 describes the implications for the Service Framework that can be drawn from the work in WP3.2. This description comprises framework structure and planned future refinements of the framework.

Section 3 addresses the actual BIONETS Service Life-Cycle and the single life-cycle activities. The chapter is introduced with a general overview and describes subsequently Initial Service Creation, Service Transformation, Service Evaluation and Selection, Service Deployment, Service Execution, and Service Deprecation in more detail.

Section 4 introduces the notion of loosely coupled interaction with regard to the Interaction Framework, a part of the BIONETS service architecture and includes definitions for two service related applications of the service architecture, namely Service Capability Acquisition and an approach for provisioning information about the user and the environment context.

Section 5 deals with the aspects of Self-Management in face of the BIONETS Service Life-Cycle. For this purpose Autonomic Service Life-Cycle Management and Resource Management are discussed.

Finally the content of this deliverable is concluded in section 6, while section 7 and 8 list the referenced literature and the used abbreviations, respectively.

2. Implications from the Service Architecture

2.1 Overview

The previous deliverable in the SP3, deliverable D3.1.1, presented the initial service architecture definition and identified the fundamental service requirements for the BIONETS application scenarios. The deliverable D3.1.1 also provided initial guidelines for this deliverable giving a high-level overview of the necessary system components and their interactions. This section gives an overview of the most important results of the service architecture specification research work presented in the D3.1.1 and implications of the D3.1.1 given for the D3.2.1.

One purpose of this deliverable is to refine the service architecture presented in D3.1.1 and analyse in a more detailed manner the topics related to BIONETS service life-cycle. These results will be used to revise and update the service architecture in the next revision of service architecture document D3.1.2 which provides an intermediate architecture definition and revision of the architecture requirements.

The BIONETS service architecture needs to cope with the issues such as the service management, content and terminal adaptation, service discovery and security aspects as well as the evolution of services and life-time. In addition, the service should be able to work properly on top the BIONETS network architecture. This kind of "disappearing network" environment, where availability of resources, network conditions and user requirements change dynamically and the co-operation between nodes cannot be guaranteed and disconnected operations are common, generate challenges for the service architecture.

In the following list we present the most essential BIONETS service architecture paradigms which affect the service life-cycle research issues presented in this document.

- *U-node, T-node*: The BIONETS computing environment is intended to be basically composed of two types of computing nodes, which are referred to as T-Node and U-Node. A T-Node is an abstraction of physical devices with hardly limited physical resources. In contrast, U-Nodes are abstractions for computing devices with substantial physical resources, for instance mobile user terminals, desktop computers or even home entertainment devices.
- *Layered view on services*: In the BIONETS service architecture the services are divided into different logical layers. These layers are *Data Access, Representation, Processing, Interpretation and Evolution.* Each layer provides appropriate means to control the layer below.
- *Technical Service Definition:* A service is a self-contained component with a well-defined functionality. Services are invoked, configured, and managed through uniform interfaces. Each service comprises a set of operations and parameters, while the operations cover the actual service logic and parameters allow configuring the behaviour of the service logic in a fine grained manner (e.g. timeouts, buffer sizes, number of parallel threads, etc.).T-Nodes and U-Nodes provide the services and allow accessing their interfaces.
- *Service Types*: We distinguish between three types of services, called Fixed Services, Migrating Services, and Composed Services. All services types are interfaced in the same way, but differ in their behaviour and scope.
- *User-Driven Service Generation:* Services are supposed to come up as solutions for complex user tasks. The User Request contains a formal description of the goal the user wants to achieve with the support of the computing environment. The Generic

Service Request is the counterpart of the User Request and describes a technical approach to achieve the goal formalized in the User Request.

The following section 2.2 discusses generally service framework principles presented in deliverable 3.1.1. Also the autonomy of a node/service is briefly discussed in section 2.2. In this deliverable the service autonomy is mostly in node level because of the technical restrictions, but the final division of whether the service autonomy is implemented in the node- or service-level is still open issue. New bio-inspired solutions and technologies researched in the SP2 could also enable to integrate the autonomy to the service level, which in turn would enable wider role and mobility for BIONETS services and also minimize the middleware type approach from the BIONETS service architecture. Nevertheless, both approaches are good solutions to certain use cases and therefore the final decision to the location of service autonomy logic is dependent which of the scenarios presented in D3.1.1 is researched

2.2 Service Framework Principles

Self-control and self-management were initially outlined as features of the service execution environment. Thus, each node is supposed to implement certain core functionality to control the service life cycle activities and provide an interface to the underlying physical system on the one hand and to the user on the other hand. For future refinements of the service framework we want to investigate how management tasks can be outsourced to service themselves as depicted in Figure 2-1. That way we intend to address the evolvement of the basic framework features in the future. However, the following sections will outline the current picture of the service framework we base our first considerations about the BIONETS service life cycle on. As the service framework definition is part of BIONETS WP3.1, this document only addresses life-cycle related aspects of the service framework. Future refinement of these aspects will be part of the work in WP3.1 and therefore covered by deliverable D3.1.2.

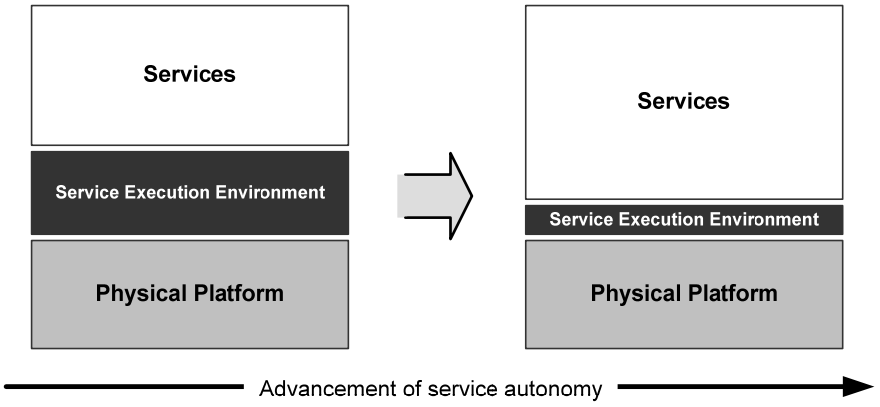

Figure 2-1: Service autonomy integration proceeding

The principles of the service architecture introduced in D3.1.1 are built on the common service pattern. Thus, in general it can be distinguished between service consuming and service providing entities. The service consumers access the services offered by the service providers when needed. Each service may be offered by multiple service providers in parallel. That way, the service consumer may choose a service provider first when the required service needs to be executed. This very special characteristic is referred to as *loose coupling* in the scope of service-oriented architectures. However, since we started to sketch service creation as a matter of composing simple service to more complex ones and service evolution as dynamic modification and recombination of composed services, we need to consider ensembles of services and accordingly service providers.

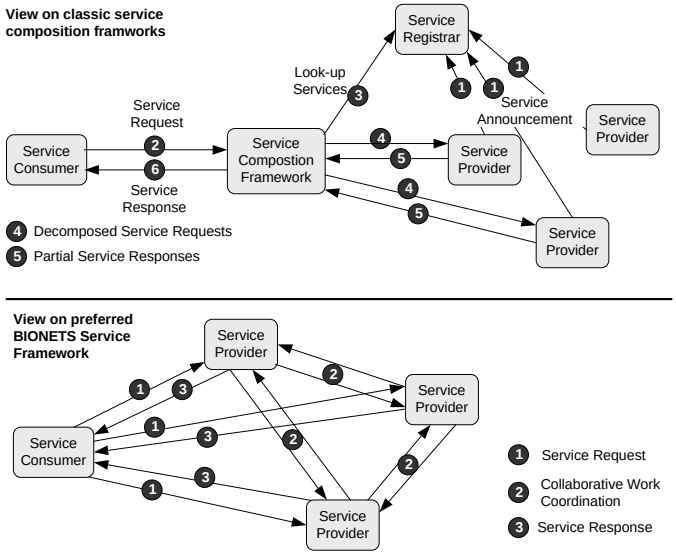

Figure 2-2: Service composition frameworks

To address service requests in those systems efforts are required in gathering knowledge about the available services and forming reasonable compositions. In classic approaches central components were introduced for this purpose, as strongly simplified depicted in the upper part of Figure 2-2. These components act as proxies on behalf of the service consumer and look up, plan, schedule, and execute an ensemble of services from several service providers that answers the service request correctly. The lower part of Figure 2-2 illustrates again in a very simplified manner the kind of composition framework we propose for the BIONETS service architecture. In general, the BIONETS approach assumes highly dynamic networks built upon groups of mobile nodes, which are additionally supposed to be mostly disconnected from superior network infrastructures. Thus, rarely spread composition engines are only of little use in this context. Instead, we will need to enable each node – at least each U-Node – to play a major role in the service life cycle management, i.e. tasks such as evolution, execution of services. These considerations become even more important when expecting some nodes to be faulty or acting maliciously. In this case cooperative processes are required to identify and compensate sources of error.

For this purpose autonomy in terms of management and control processed needs to be integrated in the service framework. We see two approaches for this goal, 1) integrating autonomy on the node level or 2) integrating autonomy on the service level. The latter approach would require making each service partially intelligent, i.e. to duplicate management functionality with each new service on a node and the respective resource consumption. Moreover, this approach would imply that strictly seen, service provision is a matter of U-Nodes only since most T-Nodes are supposed to be limited in their processing capabilities and not allowed to communicate among each other. For these reasons we focused on the integration of autonomy on the node level first. Each Node, in particular each U-Node, will provide several entities that addresses the aspects of autonomic life cycle management on behalf of the services. Inspired by the service pattern we first introduce two management elements, a Service Mediator and a Request Mediator. Figure 2-3 illustrates how these elements integrate with the principles brought up in BIONETS D3.1.1. First, each node is supposed to provide a several services, which have a classic appearance from an interface point of view. Hence, they are supposed to have a description that outlines their interface as well as functionality and they are not starting any process until receiving appropriate external events, e.g. service requests. The Service Mediators of the U-Nodes read the descriptions of the services and use the obtained knowledge for the cooperation among each other. Service Mediators are responsible for the provision of services and accordingly for answering service requests. T-Nodes are not supposed to host Service Mediators. Thus, Service Mediators of U-Nodes are required to manage the services of T-Nodes nearby also.

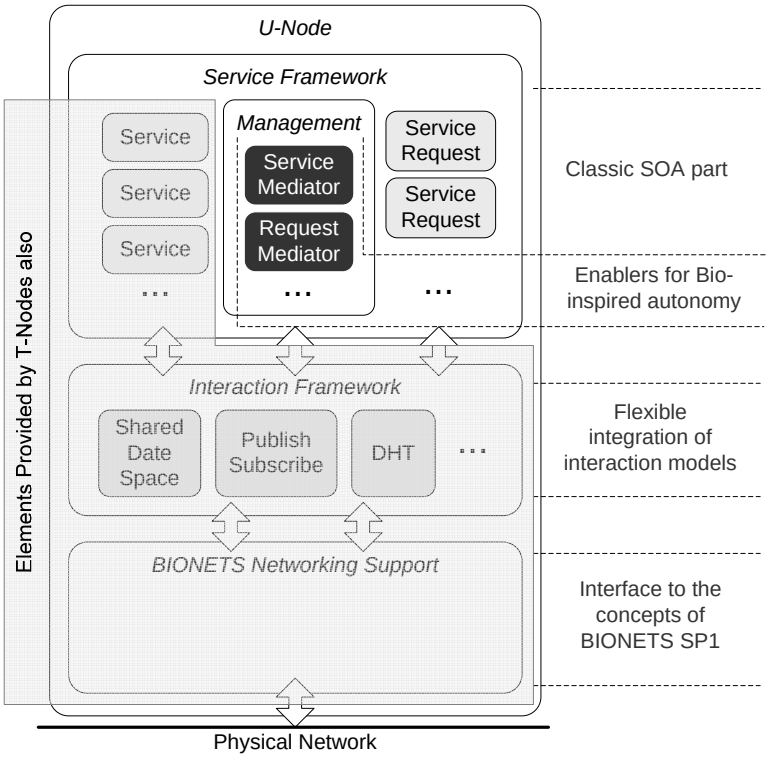

Figure 2-3: Abstract view on the envisioned Service Framework

The Request Mediator represents the bridge to the user. It creates Service Requests from User Requests while mapping human-centric problem descriptions to service-centric problem descriptions and interweaving user preferences and context information. Moreover, the Request Mediator monitors the processing of its Service Requests by the Service Mediators of other U-Nodes in the current environment, escalates relevant information to the user and retrieves user feedback for the cooperative processes.

The Interaction Framework was introduced in D3.1.1 and provides a more application-oriented abstraction of the data exchange among several nodes. For that purpose the framework provides few atomic instructions to manipulate resources on nodes. These instructions initially comprise Create, Read, Update, and Delete (CRUD). Based on these instructions interaction models, e.g., a publish/subscribe model and a data space model can be built. Hence, the Interaction Framework is supposed to enable the fast and easy creation of new interaction models for the service architecture. An interaction model defines roles for the interacting parties, principles of data exchange and data representation. The introduction of the Interaction Framework as intermediate layer is supposed to simplify the integration and evaluation of most different models for communication and coordination among nodes. Moreover, this approach benefits the emergence of node interaction driven by the service layer. More detailed information on this topic may be found later on.

3. BIONETS Service Life Cycle

3.1 Overview on the Bio-inspired Service Life Cycle

The BIONETS service life-cycle is designed to allow services to emerge in an autonomic fashion in order to gain best support for user tasks while reducing the user's efforts in service creation. This goal is regarded against the background of a pervasive service environment, composed out of services originating form the user's computing devices and those relevant in the user's context and his/her direct surrounding. In general the service life-cycle is supposed to address two facets of service emergence:

1) **The creation of service on demand of the user.**
2) **The undirected service creation to achieve completely new functionalities so far unknown to the user.**

In this section we will first only regard the service creation on demand of the user. There is a large gap between the formalization of the user's needs and formal descriptions of functions provided by computing devices. When formulating the goal for a service execution the user utilizes concepts describing the problem from the human point of view. A user who requests support for getting from one point in the city to another one does not reflect about the steps required by the computing environment to support him/her with this service. In contrast, the functions provided by computing devices in the user's environment are very special and their description may be far apart from the user's notion. Those functions may for instance resolve the user's current location, render street maps as images or display images on screens. As the given example suggests, fulfilling a user's request can be regarded as a matter of combining basic functional elements provided by the computing environment in an intelligent fashion. Consequently, the first iteration our research on BIONETS service life cycle addresses the creation of services through the autonomic and evolvable composition of services.

To ease descriptions we henceforth distinguish between two basic structures, life cycle activities are operating on. A basic building block for services is referred to as a Service Cell. Compositions of Service Cells are referred to as Service Individual. Service Individuals may also be composed of other Service Individuals in a recursive way. However, completely decomposing Service Individuals result in a set of Service Cells, which for the moment cannot be broken into constituent parts. And consequently, our research addresses the autonomic composition of services and the modification of service compositions by the use of evolutionary concepts. Service Individuals and Service Cells are supposed to be interfaced in the same way. Hence, from an external point of view they appear in the same way. So, whenever meaning either a Service Individuals or Service Cell we will use the more general term Service.

The Service Individual is expressed with a graph representation, wherein nodes represent types of Services (the elements of the composition); edges (links) represent the interactions (communication) between two Services. Two sets of edges over the same set of nodes are considered:

- Functional edges: represent the flow of data and control interactions between two Services, connecting the output of one Service with the pre-conditions of the other: an edge of the form "a->b" means that "a" is the producer and "b" is the consumer, i.e, "b" uses the output of "a. The edge is annotated with the information that flows from "a" to "b".
- Non-functional edges: represent the fitness evaluation flow between a service and its clients: an edge of the form "c->s" means that "c" is a client of "s"; it evaluates the behavior of "s" at regular intervals and sends the resulting fitness feedback information to "s". The edge is annotated with the fitness value that flows from "c" to "s".

These two disjoint sets of edges form two interactions graphs that overlap at the nodes but not at the edges:

- Functional graph: represent the overall flow of data and control information among all constituting elements (Services) of a given Service Individual.
- Fitness graph, or non-functional graph: represents the overall fitness evaluation flow from the top level client (the user of the Service) down to its constituent elements.

3.1.1 Life-Cycle Activities

The BIONETS service life cycle addresses for the beginning two major objectives, the creation of service on demand of the user and the continuous adjustment of services to cope with changing requirements in the computing environment. While the first objective addresses basics to achieve an applicable service environment at all, the second objective addresses the progressive improvement and adaptation of services in the presence of highly dynamic computing infrastructures. According to the objectives we defined three phases for the life-cycle as depicted in Figure 3-1, *Preparation and Creation*, *Evolution* and *Integration*. The Preparation and Creation phase comprises activities around acquisition of user needs and environment context as well as the creation of functionally appropriate Services addressing. The Evolution phase describes all activities concerning improvement, adaptation and pro-active creation of Services. Last but not least, the activities of Integration phase address the actual handling of Services in BIONETS environment. These phases are briefly outlined in the following sections.

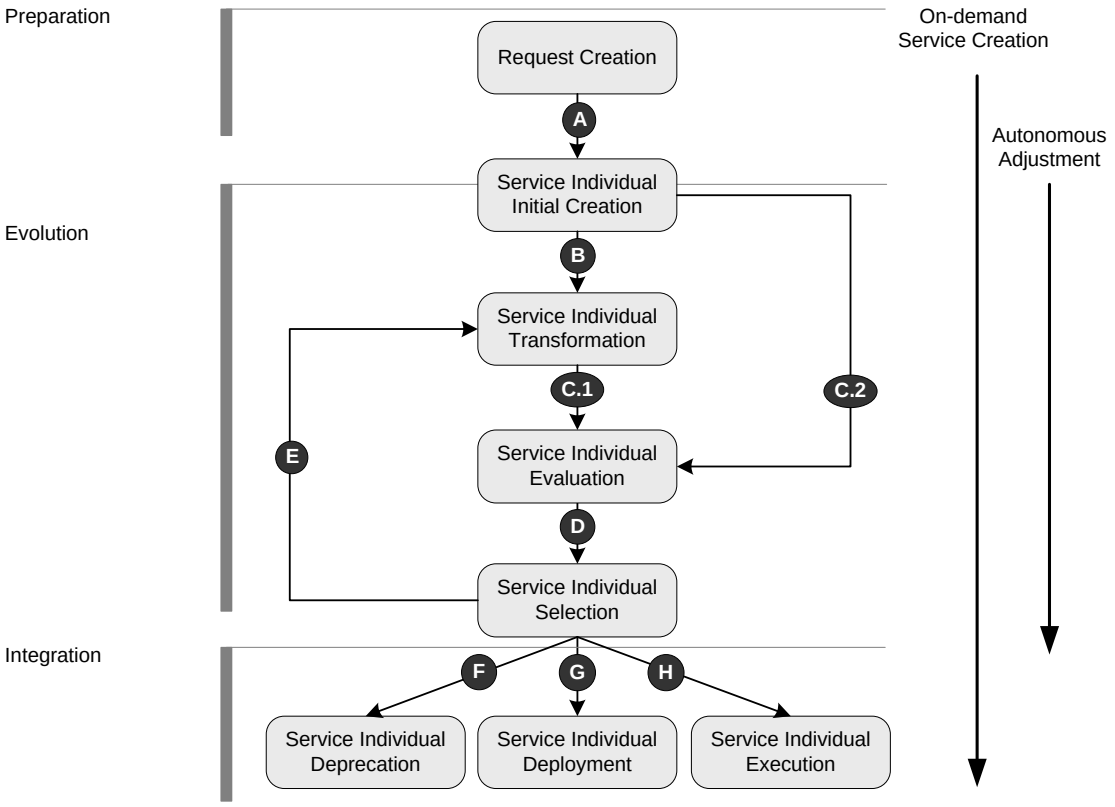

Figure 3-1: Life-cycle phases and activities

As depicted by the arrows in Figure 3-1 the BIONETS service life-cycle incorporates a complex flow of data between the single activities. The single data exchange interfaces between the life-cycle activities are marked with reference point labeled with capital letters from A to H. Table 3-1 shows the names and explains the data exchanged at the reference points in brief.

Ref. Pt.	Description
A	Service Request - a formalized goal description from the users point of view
B	Service Individuals fulfilling at least the functional requirements defined in the Service Request and ready for transformation (i.e. improvement)
C	Service Individuals supposed to be evaluated
D	Tuples of Service Individuals, corresponding bindings and performance evaluations
E	Service Individuals to be transformed again
F	References to Service Individuals that should be deprecated
G	Service Individuals to be deployed to a destination environment
H	Service Individuals that are supposed to be executed on-the-fly

Table 3-1: Life-cycle reference points

3.1.1.1 Preparation and Creation

The Preparation phase initially comprises the *Request Creation* and partially *Service Individual Initial Creation.* Since we regard on-demand service creation, the *Service Request* represents the major orientation for the life-cycle activities when compiling a suitable service matching the users need. The *Request Creation* addresses the creation of a formal description of a user task while incorporating also information about the context (i.e. current situation, user preferences, etc.). The *Service Individual Initial Creation* (SI-InCrea) is intended for the creation of Service Individuals that suffice the Service Request from the functional point of view.

This general definition of both activities allows us to investigate very different approaches for the reflection of user goals and the respective creation of Service Individuals, i.e. composition of Service Cells. We argue that it is possible to distinguish these approaches according to the source of knowledge about the Service semantics, as having this knowledge is the fundament for composing Services in a reasonable way. For the moment we see three possible sources for this knowledge, the *user*, the *service developer*, and the *computing system*. In the first case the user composes service in a manual fashion while defining order, conditions, control/signaling and data flow for the execution of the single constituting services. Hence, the user utilizes own knowledge to add value to the service environment by creating new services. Although the approach of user-generated functionality is charming, the applicability decreases with the service complexity. While simple problems can be addresses with simple service compositions, finding service compositions for the solution of complex problems may be very challenging for the user.

When regarding the service developer as source of knowledge, we are talking about knowledge that is delivered together with the service implementation. Thus, the Service itself is extensively described with meta-information, which comprises descriptions of interface semantics and functional semantics. This idea is for instance reflected in WSMO [84], SWSO [85] and, OWL-S [86] semantic description languages for the Web Service architecture. Semantic meta-information can be utilized by automatic composition processes to find a reasonable order for the execution of services and the data flow among them. Hence, in this approach the user is not required to define the composition of service himself/herself, he/she just needs to define the goals to be achieved, i.e., the problem but not the solution. Although automatic service composition based on semantic service descriptions is a theoretically promising and partially practically proofed concept, there is a trade-off between the required service development efforts and the actual applicability in large-scale open systems.

As third source of knowledge about service semantics we identify the computing system itself. Instead of assuming service to be annotated with extensive semantic descriptions the

service environment is supposed to 'learn' how certain services can be composed. For that purpose an initial description of user goals and successive user feedback can be utilized. Few pre-defined heuristics could additionally help to avoid critical malfunctions. While positive examples for self-learning systems are currently only known from other domains, respective features would perfectly fit into the notion of self-evolving services. However, in this deliverable we will first concentrate on the definition of a solid base for all life-cycle activities and therefore start with an approach of service creation which depends on the users' knowledge about service semantics.

3.1.1.2 Evolution

The Evolution Phase comprises Service Individual Transformation (SI-Trans), Service Individual Evaluation (SI-Eval) and Service Individual Selection (SI-Sel). Based on several Service Individuals SI-Trans is supposed to form new variations of Service Individuals with slightly differing compositions and configuration. Additionally, the Service Cells composing the Service Individuals may be transformed on code-level to achieve a high diversity in the service variation. For the transformation of both, Service Individuals and Service Cells we envision the utilization of methods taken from genetic programming, e.g., mutation and cross-over. SI-Trans is supposed to be an ongoing processes in order to find the Service Individuals that best address functional and non-functional requirements of the Service Request.

A major aspect of systems inspired by nature is the self-control. With regard to the life-cycle, self-control addresses the autonomous evaluation and selection of Service Individuals. SI-Trans may create numerous Service Individuals, which need to be compared to each other in order to find the one that best suffices the Service Request. The intention of SI-Eval is to estimate or test the performance of Service Individuals in the destination environment, at least under simulation conditions. However, the evaluation of Service Individuals in a certain computing environment is not only about the composition plan, but also addresses instances of this plan, i.e. sets of actual Service Cells that substitute the nodes in the composition graph while implementing the denoted type of Service. Consequently we expect that for each Service Individual there exist multiple disjoint sets of Service Cells that represent instances of the Service Individual. During SI-Sel each Service Individual needs to be regarded together with its possible bindings in the destination environment. Hence, Service Individuals should not only be functionally correct, but effective with regard to various criteria like required calculation time, battery power, storage consumption etc. Within the BIONETS project a system is regarded whose functionality has not to be learned by the user but which gets to know the user self-reliantly. Therefore, the user should not define preferences explicitly on her/his own, but the system should learn the user preferences with regard to the current context by means of user feedback. The performance of the Service Individuals already estimated to provide the requested functionality has to be evaluated according to these given criteria. Notably, since the given criteria could compete among and influence each other, the evaluation of the SI fitness has to be considered as a multi-criterion optimization problem.

3.1.1.3 Integration

The fourth phase of BIONETS service life-cycle aims at the maintenance of services in the computing environment. For that purpose we consider three important activities: Service Individual Execution (SI-Exec), Service Individual Deployment (SI-Deploy), and Service Individual Deprecation (SI-Depr). SI-Exec simply addresses the execution of a previously created and positively evaluated Service Individuals in the current computing environment. SI-Deploy generally concerns the propagation of Service Individuals to other system entities, in order to make them available to clients of these entities. In particular we will differentiate between a propagation of Service Individuals among several nodes and propagation across several functional layers within one node. SI-Depr consequently represents the counterpart to SI-Deploy. Since Service Individuals are supposed to be successively replaced by better ones and the user's needs for certain services may elapse, SI-Depr describes the process of removing Service Individuals from the system.

3.1.2 Life-Cycle of Primary Service Cells

In the BIONETS service environment the services can be divided in three categories: *Fixed Services*, *Migrating Services* and *Composed services*. Fixed Services abstract the functionality which is special to certain node, i.e. usually some kind of the features of physical devices abstracted by these nodes. As the name suggests Migrating Services are services, which are not bounded to a certain node and their implementation/functionality can migrate between capable nodes, i.e., the code of the service implementation, as well as the execution state could be moved between different nodes. The third service type Composed service a composition of other providing certain functionality to a user or another service. The Composed services are created in the BIONETS environment in an autonomous or "semi"-autonomous manner based on the user needs or context information, but Fixed Services and Migrating Services are a kind of primary service, quite similar to the traditional services in the current systems, and they need to be initially "programmed" by software developers and then injected to the BIONETS environment. These primary services are called *Primary Service cells* and they are the basic building blocks of the BIONETS service environment. These Primary Service Cells can be regarded as "primeval" service components, underlying classic service life-cycle principles, since they are not results of service variations but initially coded functionalities. The Figure 3-2 below illustrates the different stages in the traditional life-cycle [87][88] of Primary Service Cells.

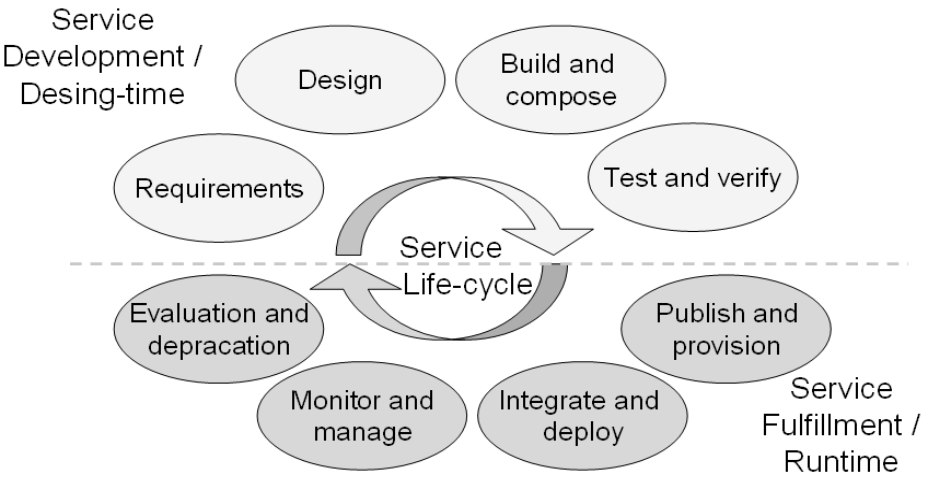

Figure 3-2: Traditional life-cycle of Primary Service Cell

The ability to effectively manage the lifecycle of services is fundamental to achieving success within a BIONETS service environment. In the Figure 3-2 we can see that the life-cycle of Primary Service Cells is divided into two broad categories; Service Development/Design-time and Service Fulfillment/Runtime. Design-time aspects of this type of management include such areas as service requirements analysis (Requirements), design and modeling a phase (Design), actual code-implementation phase (Build and compose) and finally phase where the designed and implemented services are tested and verified to make sure that they will meet the requirements of the BIONETS environment. All of these aspects are very similar to the "normal" service development done outside of BIONETS environment and the life-cycle of BIONETS Primary Service Cells is not that much different that with traditional current services. Of course in all the design and implementation phases the special characteristics of BIONETS service environment need to be taken into account (e.g. network connectivity, service evolution, etc.)

The run-time aspects of the BIONETS Primary Service Cell lifecycle include publishing and provisioning the service (Publish and provision), deploying the service to the target node and integrating it into composite applications (Integrate and deploy), monitoring and managing its usage (Monitor and manage), and evaluating the functionality of the service and if needed the deprecation of the service (Evaluation and deprecation). Although the basic runtime phases of the BIONETS Primary Service Cells are not that different form traditional services the actual

implementation of these phases is quite different. In the BIONETS service environment even the Primary Service Cells may utilize bio-inspired functionalities, such as evolution and self-management.

Although Primary Service Cells are autonomic units and fully capable to offer services to users and other services, their main purpose in the BIONETS environment is to be building blocks of Service Individuals. The life-cycle and the whole functionality between Primary Service Cells and Service Individuals is quite different and the Service Individual life-cycle related issues are discussed in following section 3.1.3 Life-Cycle Activities.

3.2 User-driven Service Request Creation

In the BIONETS project a user request should contain a formal description of the goal the user wants to achieve with the support of the computing environment. Moreover a generic service request describes the technical approach to achieve the goal specifying how several functional elements (i.e. Service Cell) need to be arranged to fulfill the user's needs [81].

Basically from our point of view this could be done with different tools used for formalizing the user request but leading to the same formal service representation. The first approach is a user oriented service creation process where an intuitive graphical notation could let users express their own service needs assisted by suitable tools. The result will be an internal representation (e.g. XML-based) derived form the graphical one. The same goal could be achieved through a semantically enhanced approach based on the information the U-Node could collect on the user e.g. user profile, user's location etc.

This last point could be a matter of investigation and study during the ongoing BIONETS project.

User-driven request creation is thus expected to provide a manual creation method for service requests. As generally introduced in section 2.2, it is aimed at providing working versions first and integrating a higher level of autonomy in a refinement step. It is thus envisioned to eventually develop an automatic mechanism for transforming user requests posed in a natural language into services requests serving as plans for services compositions able to match the user's request. These transformation procedure would enable users to specify *what* they want to achieve instead of *how*, meeting one of the main objectives of service oriented architectures (SOA).

3.2.1 Approach to Service request creation

The approach to service request creation is to let users express their own service needs through a graphical representation whose definition will be assisted by suitable tools.

The main idea behind personal service creation is to shorten the analysis/development/marketing chain, giving users the possibility to satisfy their specific needs, creating new services by means of an assisted assembly process.

This approach poses a number of requirements, both on the graphical representation side and on the definition of the tool.

3.2.1.1 Requirements for the Service Request Creation language

The Service Request Creation language should be simple enough to be used by users with minimal technical skills. The approach is thus driven by the goal of making service composition as intuitive as possible.

We suggest using a graphical notation supporting a composition paradigm based on basic Service Cells. We are thinking of a simplified graphical notation inspired to existing service composition approaches (e.g., Active Bpel Designer [90], Eclipse Bpel [91]) with the goal to have service definitions described using this simple graphical notation must be intuitive and self-explaining.

With this rationale, the fact that probably an end user is not familiar with the concepts of operations and input and output parameters, leads to the idea of proposing a similar but more familiar concept of Service Cell configuration to customize its behavior.

The same rationale drives the definition of the constructs for specifying service control flow: for this purpose we define the notion of notifications linked to actions instead of the usual programming approach of operation invocations. We think that this event driven approach can be easily understood and represented in a sentence pattern like "*on* <event> *do* <action>" very close to JAIN SLEE event definition where a Service Building Block is a software component that sends and receives events and performs computational logic based on the receipt of events and its current state [89].

The description of each Service Cell can be defined by the following features:

Concept	**Description**
Generalities	Name, version, description, type
Configuration attributes	Parameters of the cell that specialize its behavior. Can be mandatory or optional
Variable attributes	Public variables exposed by the Service Cell.
Notifications	Represent the events a Service Cell can raise.
Actions	Represent the operations exposed by the Service Cell, i.e. the actions it can perform when an event notification is received. Actions represent the internal behavior of the Service Cell and, as they are basic and atomic pieces of processing, their definition is not to be detailed further.

The definition of the Service Individual is achieved by composing basic Service Cells according to the event driven control flow specification approach stated above. In other words, a Service Individual is a set of Service Cells, that will be chained based on the events they can generate and handle ("*on* <event> *do* <action>").

3.2.1.2 Requirements for the Service Request Creation Environment

The service request creation tool has to support the requirements defined above. Besides, it has to assist the user in the definition of syntactically and semantically correct service definitions.

While a syntactic check of the composition can be easily implemented, a service creation tool for end users could benefit from a semantic description of the Service Cell to suggest meaningful connections.

Attention has to be put on guiding the configuration process of each Service Cell, in order to enforce privacy, billing and security constraints (e.g. we try to avoid anonymous access to services). For this purpose some values in the configurations of Service Cells could be enforced to be bound to user's profile values.

The graphical notation and the compositional approach will require no software development skill to develop a service. Since the graphical notation has to be intuitive and self-explaining, using the tool should take a short learning curve, i.e. it should be possible to learn the tool hands-on, just by going through a few examples or using it. As for the implementation, the tool has to be lightweight, i.e. software requirements on the user device have to be limited to having an HTML browser installed and no more.

One benefit of this choice is to allow the service definition to be accessible through different terminals and from different locations: i.e. the user should be able to access its service definitions either from its computer at home or at the office and he should be able to handle and modify them either from a laptop or from a hand-held device. This also means that the same service definition has to be presented to the user according to the capabilities of the

device he is using, e.g. if he edits the service definition from his laptop, then he can benefit of a full graphical environment, if he accesses the same service definition from his PDA, he will be presented with a simplified interface, possibly supporting a restricted set of editing operations.

3.2.1.3 Inside the service request creation environment: service descriptors

From an architectural perspective, the service request creation environment is the front-end needed to create new service according to the users' needs. But service definitions need an execution environment to be delivered. To allow the maximum flexibility to connect different service creation front ends to different target execution environments, we need to define an intermediate service request representation which we will call service descriptor.

A service descriptor will fully represent the semantics of the service in a general way, so that both the graphical and the executable representation can be easily derived from it. For instance, this internal representation could be XML-based, so that transformations into any other representation can be easily achieved using well known techniques (e.g. XSLT transformations).

This approach allows keeping the service creation environment decoupled from the runtime environment, thus promoting reuse and keeping it open to different back-end implementations.

In case of an autonomic service execution environment, such representation could be collected as a snapshot of the actual running service. Possibly it could be used as the starting point to go back in the service creation chain to provide the user with a human readable representation of the evolved service.

3.3 Initial Service Creation

In the following one possible way of implementing autonomous service creation is outlined, focusing on a component based approach; other approaches such as state-based or semantically enhanced ones are assumed to be topics for future investigations.

As introduced in the preceding section, user needs and preferences have driven the creation of a service request providing a service descriptor. From such a service descriptor, the problem is now to generate a concrete, mostly executable representation of the service.

The descriptor is assumed to be a XML-based representation of a set of inner services that must be invoked: those services are either primitive services named Service Cells in the sequel, or are themselves the result of the composition of some other services considered as Service Individuals. In other words, the BIONETS service model we have in mind is hierarchical and recursive. Moreover, each service may require the call to other services mentioned in the service description in order to realize its function (i.e. the notifications that a service may raise correspond to the actions it requires other services to execute so that it can fulfill its assigned role). Consequently, each service that is part of the service description provides a set of functionalities, and, for this, uses functionalities offered by some other services. A convenient way to represent such dependencies is to rely on a software component-oriented approach: dependencies between services are represented as bindings between use-provide ports.

An Architecture Description Language (ADL) is a XML-based specific syntax that is recognized to be a suitable way to define a software component based application [77]. Such a language will be defined to support BIONETS's services initial creation, considered as a variation or extension of the XML-based service descriptor one. The purpose is that the service initial creation process will instantiate a first Service Individual by parsing a document written in this language. The result of the parsing will be, depending on the provided document information:

- The discovery of some Service Cells or Service Individuals references for services that are already locally executing or locally managed by the U-node.

- In case a Service Cell must run exclusively on behalf of the specific Service Individual being instantiated, then, this cell is also instantiated, locally on the U-node.
- The intention to connect/bind to some remote Service Cells or Service Individuals, that will be later discovered on the BIONETS.. One important fact here, contrary to classical software component approaches, is that the connection between remote entities will not need to be solved before the execution of a Service Individual can start, nor be made explicit even at runtime (see next bullet).
- The connections between all such cells according to the BIONETS operational model (see Figure 3-3):
    - effective connections (through standard service referencing) between services managed locally,
    - virtual connections to services that are remote and that may even be kept unknown to the caller service: indeed, from the service caller point of view, only the reply of a service invocation is important, not the identity of which specific Service Cell or Service Individual has produced the reply.

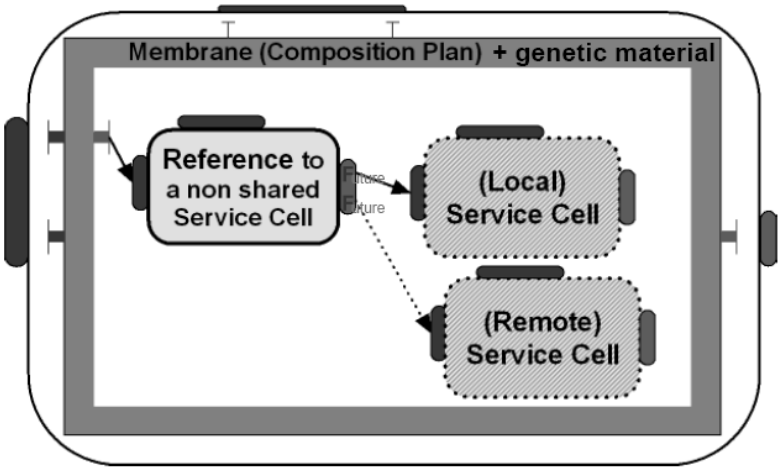

Figure 3-3: Schematic representation of a BIONETS Service Individual: on this example, it is only composed of Service Cells

In the BIONETS context, the invocation of a service relies on a very loose notion of service connection: connection is totally asynchronous, and only the reply of the service invocation is important for the caller. This implies that even the identity of the Service Individual or Service Cells that produced the reply can be kept unknown to the caller. Of course, this feature is not necessary between Service Cells or Service Individuals that have been created and deployed on the same device. Such a very loosely-coupled communication mode can be managed in the functional code by having service replies be turned into first class entities known as futures: a future is an entity that represents the promise of a reply, and which is subject to a transparent wait by necessity [79] and transparent update from the underlying interaction framework (see section 4.1).

By relying on such a hierarchical vision, as pioneered in the software component context by the Fractal component model [78], each Service Individual can be associated to a self-contained software entity. In particular, this entity contains a membrane, which can support all the interactions with the external environment, in particular, with other self-contained Service Individuals of the same type with which mating may arise. More generally in this vision, all non-functional aspects associated to a Service Individual can be well and cleanly separated from the functional logic as they are described in the membrane. Referring to the next section, the two graphs that respectively represent the functional organization and the fitness evaluation can be stored within the membrane.

Parsing the ADL we can first generate a purely local representation of the Service Individual: it is both a local and possibly simplified version in the sense that all included Service Individuals or Service Cells that are assumed to be found remotely, will not be really instanciated but emulated locally according to their specification, and what we could know about their implementation characteristics. As the reference point C2 in Figure 3-1 points it, the purpose of this simplified Service Individual instanciation is to try to evaluate the expected Service Individual behavior and performance (including its Quality of Service) before its deployment. If needed, it can be adapted and improved, by successive -- possibly random -- transformations (point B), before being effectively deployed on the BIONETS environment (point G). Those possible transformations include the possibility to replace a Service Cell by any other, as soon as they provide a similar semantic, even if it is not of same type.

It is worth mentioning that this way of evaluating and improving a Service Individual before letting it run can also serve to explore less user-driven service compositions. Indeed, we also intend to explore some automatic and autonomic generation and composition of services during the project. Assuming a semantic service description is provided as a user service request, the environment will infer a service description in a totally automatic way. This means that even an ADL like description will not be produced (at point A). On the contrary, the service initial creation will start from a mostly empty Service Individual, whose membrane will initially only contain the overall goal. The loop expressed by points C-B-H will enable to refine this individual according to this goal, producing an executable service version as exemplified on figure 4.2. The refinement could itself be based on some heuristics, or even some previously learned suitable compositions (e.g. according to the problem domains).

3.4 Service Transformation

The online evolution of services as proposed by BIONETS requires that transformations be performed on selected services in order to obtain a new generation of services which may be better than their parents, so that evolution can proceed. These transformations are the source of variability in the system, which makes evolution possible.

Not all of the service implementation or its constituent building blocks may be allowed to evolve. Some parts may be fixed, not subject to evolution. This is related to the roadmap outlined in D2.2.2 [80]: we may start by evolving service parameters only, then combining service building blocks, and ultimately eventually evolving the implementation of the building blocks themselves at a finer (programming language level) granularity. The part of the service that is allowed to evolve is specified in the service genotype.

The service genotype is the part of the service that can be transformed for evolutionary purposes, i.e., for producing the next generation of services. As in biology, the genotype of a parent service is inherited by its offspring. In order to obtain such effect, an evolvable service must be designed such that the evolvable part is included in the genotype. This is done via normal (human-based) software engineering and implementation. Therefore the system is not able to make legacy software evolve, since this has no visible genotype that can be manipulated.

Service Transformations operate changes in the genotype in order to produce a new generation of services. According to the roadmap of D2.2.2, transformations are envisaged at the following granularities:

- *Parameter granularity:* Parameter configuration and optimization: at this granularity, only the service input and control parameters can be modified; the service internal structure and implementation do not change.
- *Component granularity*: Combination of Service Cells: at this level of granularity, transformations operate on the internal composition of Service Individuals, by adding, removing or modifying its constituent components.

- *Code granularity*: Code modification: at this level, the actual implementation of internal algorithms of Service Cells may suffer partial changes.

As discussed in the roadmap of D2.2.2, these progressive granularity levels imply increasing difficulty and risk, and therefore pose increasingly challenging research issues.

The first level (parameter) is strictly speaking not really evolving new functionality, as only fixed parameters may change, so it remains at the self-configuration and self-optimization level. However, in practice it is already quite challenging to obtain the optimum combination of parameters in a complex scenario with multiple services and heterogeneous disconnected network operation, so its value cannot be dismissed.

The third level (code) is the most challenging one, as modifying running code in a non-disruptive way is currently an open research problem. This level will be covered in WP2.2 Task 2.2.3, which will use a gene expression model for code generation and regulation: code written in a chemical language may occur at varying concentrations in the system, and the system regulates these concentrations to achieve the desired behavior, guided by reward/punish fitness feedback signals coming from the clients. Since transformations then affect only one instance of a service (and not all the individuals present at large concentrations) one can expect that the impact of harmful transformations can be minimized, while at the same time allowing favorable mutations to spread in the population. This research is currently starting, and the results will be reported within WP2.2. In the present document we focus on the second level of granularity, i.e. the level of the component combinations that produce a service.

We now look at how transformations can be performed on the graph representations constituting the Service Individuals, as described in section 3.1. As a first observation, we can now reduce transformations at parameter granularity to modifications on the weights annotated on the incoming edges of the functional graph of the corresponding service. Component granularity transformations now correspond to graph manipulations that change the shape of the graph, i.e. by adding or deleting nodes and links. Code transformations would correspond to modifying nodes in the graph (i.e. modifying the internal implementation of a service). If a Service Individual is a composite one, i.e. made up of several internal components, then component-level transformations can also be applied there. In the case of an indivisible Service Cell, the only possible transformation of its internal structure is at code-level. Code-level transformations are left for WP2.2 Task 2.2.3, and therefore will not be discussed any longer in this document.

Transformations may be of two types:

- Mutation: a transformation that involves only the concerned Service Individual, by changing its internal graph structure.
- Recombination: a transformation that involves more than one individual, for example, by combining portions of different graphs to produce a new individual. Crossover is a special form of recombination in which two individuals interchange parts of their genotypes.

Transformations on Service Individuals, especially recombinations, could be aided by the service mediators: it could help finding compatible individuals belonging to the same species, and with some desirable traits, for instance, an individual that has a high fitness for the given service environment could represent a good candidate for “mating”. Service mediators then provide an infrastructure for discovery mechanisms in order to find these mating candidates.

Before any transformation can take place one must identify the portion of the graph that corresponds to the genotype, i.e. which can be subject to transformation. This could be done by marking the nodes and links with a probability of change (if zero then it is not part of the genotype).

Transformations concern the functional graph only. However, the fitness graph will also change accordingly, since the new service might result in a different fitness.

An issue in fitness evaluation in this graph model is the impact of the fitness of the Service Individual as a whole on the fitness of its internal components. Somehow the fitness reward value should be split among these internal components, in proportion to their own sub-fitness. Composite individual must then embed an algorithm that splits this reward accordingly. A concern is when a service is distributed through several nodes, such as a multiparty conference service, or a peer-to-peer overlay routing service: one could imagine at first glance that the user who starts the system has the responsibility to control the fitness distribution procedure, however this has fault tolerance concerns in case the initiator leaves the system or becomes disconnected (common in BIONETS scenarios). Such a shared service (several users) should somehow allow fitness feedback flow in a decentralized way. This is a subject of further research.

We must stress that fitness evaluation is performed periodically, regularly, and it is valid for a limited period of time. If not evaluated, the fitness of a service “decays” with time. A first model for this was proposed in [83]. We could enhance that model such that the speed of decay depends on the service's goals and time scale of service functionality provision. Two systems with similar goal would then have a similar lifetime. This is also a current research issue.

Competition is an important element for evolution, because evolution needs choice among different variants. We must therefore consider the impact of competition on the fitness and aging factor of a service: a service that is not able to increase its fitness before it gets old ends up extinct. That’s a survival of the fittest principle. However there might be cases where non-competitive services remain in the system because they are the only alternative to support other services. In human societies this is often the case of public services, which are necessary but hardly competitive. This issue requires a liaison with WP3.3 in order to study the impact of our biologically-inspired models on actual services business models.

3.5 Service Evaluation and Selection

The concept of "fitness", intended as the ability of a service (either SC or SI) to match the features of the environment, is a central one in the BIONETS framework. The notion of fitness corresponds here - broadly speaking - to the ability of a service to achieve some target goals. The fitness can be evaluated according to measurable objective criteria (individuating some performance metrics which can be evaluated by the service client, e.g., throughput, delay, resource consumption etc.) as well as according to subjective criteria, which depend on the peculiar needs/expectations of the final user. The latter point requires the development of new methodologies for gathering feedback from the user in a non-invasive way; this point is out of the scope of the current deliverable and will be left for future work.

The service evaluation and selection tasks are made more difficult by the fact that the fitness needs to be evaluated at both the SI level (where both objective and subjective metrics can be used) as well as at the SC level (where objective performance only can be used). Further, the user is interested in SIs only, which perform the task requested, but an efficient mechanism for dynamic composition of Service Cells needs to rely on evaluation of the fitness of each single SC. At the same time, it should be kept in mind that the evaluation of SCs in isolation may result of little help for achieving the composition of an effective SI, in that it overlooks the aspect of interactions among SCs, which is expected to play a dominant role in determining the performance of the resulting SI.

As defined in Sec. 3.1.2.3, three different elements of service evaluation need to be considered:

- rating of the fitness of available SCs;
- evaluation of the ability of the current SI to fulfill the high-level goals determined by the user and translated to the service request;
- ability of the current SI to fulfill the user's preference.

Let us focus, for the moment, on the first two issues (the third one involves issues related to human-computer interactions and falls therefore out of the scope of the research to be carried out by the BIONETS consortium).

The evaluation and selection of services in BIONETS builds on the notion of fitness. For composing a SI, SCs with higher level of fitness should be used, in order to maximize the likelihood of getting a successful SI. This requires that the fitness level of SCs can be compared, i.e., each SC needs to be ranked in a consistent fashion within the connectivity scope of the service (i.e., the "connectivity island" as termed in D1.1.1).

This requires the service framework to be able to break down the global service requests into lower-level requests of services, which can be provided by single SCs, and which can then be used to build a purposeful SI.

For this purpose, we assume that each SC is associated to (i) a description of the service offered (ii) a value, in the range 0-1, describing its performance (hereafter, its fitness). Among all the SCs found in the connected island which correspond to the description (in terms of task) contained in the request sent out by the mediator, the one with the highest fitness should be selected. The whole process is very close to the functioning of a Web search engine, which has to rank all documents which match the query. The fact of selecting the one with the highest fitness corresponds to the "I'm lucky" option on Google, for example. We term therefore the name "SC-Googling" for describing the process of service selection in BIONETS. Indeed, when creating a new SI, a service discovery process is run for finding, within the connectivity scope, SCs able to perform a given task. Responses are then collected and classified according to the respective fitness level, i.e., they are ranked according to their fitness. It is worth remarking that, in algorithmic terms, the fitness evaluation process turns out to be equivalent to a mechanism for ranking objects in a distributed database.

We consider three possible methods for classifying (and therefore selecting) services. These methods are not mutually exclusive, and could be hybridized for obtaining more efficient techniques. With respect to the service life cycle diagram depicted in Figure 3-1 this fits mostly the SI_InCrea phase and the reference point "A" in Figure 3-1. The same mechanisms can be applied to the SI evolution, in order to enhance the behavior of the resulting SI.

Gossip-based fitness evaluation: Upon the reception of the list of the SCs performing the desired task (which follows the execution of the service discovery process), the requesting service issues a SC evaluation request, asking all nearby nodes to express their score (i.e., evaluated fitness) of the SCs found. Only scores received from users which are trusted are considered in the evaluation process. The average score of each available SC is computed and the one with the best evaluation is chosen. This relies on some form of "gossiping", in which the neighbors act as trusted entities which can provide feedback on the ability of a given SC to fulfill its goals. In its most simplistic form, all neighbors are assigned the same "weight" in evaluating the fitness of any SC.

Popularity-based fitness evaluation: The requesting service receives a list of available SCs in the neighborhood able to perform the requested task. SCs are classified based on their popularity, intended as the number of copies of the particular SC found in the neighborhood. Such a choice relies on the underlying assumption that the fittest services are able to replicate themselves successfully, so that the popularity of a service (intended as the amount of copies present in the neighborhood) can be considered a good indicator of the SC fitness. This mechanism relies on the idea that successfully used SCs could be cached in the local storage of U-Nodes for future use. In this way, more successful SCs have a higher chance to reproduce themselves, and to be found. This mechanism is analogous to the one used by various P2P engines for classifying the results of a search for multimedia content.

Graph-based fitness evaluation: In BIONETS, as detailed in D2.2.2, the fitness of a SC can be obtained by means of a distributed mechanism. Each SC gets a mark (indicating its evaluation by the service client) by each SI currently using it. This is described by means of the fitness graph (or non-functional graph) as described in Sec. 3.4. Let us assume that service **A** is used (and evaluated) by services **B** and **C**. Let *f(x)* denote the fitness level of service *x*,

and *e(y->z)* the evaluation of service *z* provided by service *y*. The fitness of A should therefore depend on *e(B->A)* and *e(C->A)*. On the other hand, we clearly want the evaluation offered by successful services to play a more important role in assigning a single value to ***f(A)***. We assume that such weights are provided by the fitness value associated to the service client, scaled by a factor λ. In this way, we have:

$$f(A) = \lambda \cdot \left[f(B) \cdot e(B->A) + f(C) \cdot e(C->A) \right].$$

In this way, we can define a recursive scheme, which links the fitness value of all nodes.

In order to better formalize the problem, we may consider the matrix **E**, whose (i,j)-th entry correspond to *e(j->i)*. Also, let us denote by **f** the vector of fitness values associated to the various services. It is easy to check that the recursive property defined above can be formalized as:

$$f = \lambda \cdot E \cdot f.$$

This equation does not present a solution for all values of k, but only for those k which are eigenvalues of the matrix **E**, i.e., they are solutions of

$$\det(E - \lambda I) = 0,$$

where *det()* is the determinant and **I** is the unit matrix. In mathematical terms, **f** is an eigenvector of the matrix E relative to the eigenvalue λ. In general, for a given λ, such system may present more than one solution. Following some considerations which are out of the scope of the deliverable, we can choose the eigenvector corresponding to the spectral radius of E, i.e., to the eigenvalue λ with the largest absolute value.

While this mathematical machinery may seem unnecessary to study the fitness evaluation/selection problem, it is remarkable that it is exactly the same mechanism used by Google1 to rank the results of a search [69]. In other words, SC-Googling translates into the application of a Google-like ranking mechanism to SCs. Such an application is – however - far from trivial. The difficulty comes from the fact that the system is often disconnected and evaluations are performed asynchronously [70]. We are further investigating how this can be performed, trying to generalize the results in [70] to more dynamic scenarios.

3.6 Service Deployment

The Service Deployment primarily deals with making services in an efficient way utilizable to the users of the BIONETS environment. We have identified two challenges on the way towards this goal.

First, services need to be distributed among several nodes of the BIONETS environment, to make them physically available to users by their private equipment. This is required as there is no guarantee for the lasting availability of remote access to Services in the presence of disconnected networks. Hence, to make a certain Service continuously available it may become necessary to migrate this Service to a node that can be accessed by the user directly.

Secondly, there should be an opportunity to efficiently execute services without accessing resource consumptive control and management entities if not absolutely required. This also allows us to address devices with limited physical capabilities. Currently, we are thinking about outsourcing autonomous management functionality needed be all services hosted on one node to separate components of the framework (which are present on each U-Node). Only the actual service/business logic remains in the services, while service react on demand. Thus, for the moment Service themselves are not supposed to be autonomous in terms of running

[1] In reality the mechanism used by Google to rank pages builds on a variety of mechanisms; however the key in its success and the main innovative contribution of Google with respect to its competitor at the time it was introduced was undeniably the PageRank method [69], which is analogous to the mechanism we presented.

self-contained processes and partially interacting with the outer environment in a self-determined fashion. However, self-management is introduced to cope with the worst case, but we argue that there will also be plenty of situations profiting from less constraint environment conditions. In those situations we can shortcut the management layer and establish simple, direct interactions among the services based on classic interaction patterns.

Both deployment methods address the transfer of service logic, but in an orthogonal fashion, as depicted in Figure 3-4. For that reason, we comprise them following as horizontal and vertical service deployment. In the following section outline these methods and there expected impact more in detail.

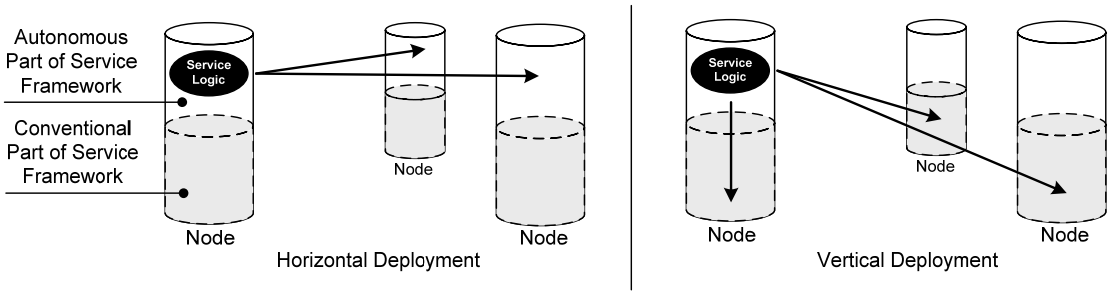

Figure 3-4: Horizontal and Vertical Deployment

3.6.1 Horizontal Service Deployment

Horizontal Service Deployment, here more general referred to as service mobility, means the mobility of the service or part of the service between different nodes in the BIONETS service environment. It is to note that not are Service Cells are supposed to be able to move between several nodes, as for instance a Service Cell may abstract functionality that can only be provided by the node originally hosting it. Service mobility can be classified into service selection, service migration, and service adaptation. Service selection is divided into two methods: First method is user active method where user selects a service himself, and second method is user passive method that provides a service according to user context by pattern recognition of for user action. In Service migration the service (code of the service implementation and the execution state) is migrated from current node to other node. In the service adaptation provides that the service is suitable for the device when attribute of previously used device and that of migrated device is different.**Error! Reference source not found.**

The BIONETS environment has to offer a scalable and effective approach for service migration. In a migration procedure a service replicates and distributes itself throughout a network automatically to another U-node. Whether the service moves just itself or replicates is application dependent choice. This operation needs also fast adaptation functionalities, so that the service can rapidly adapt itself to changing platform and environment.

In BIONETS environment there are two types of services which are capable of migrating from node to node; these are Migrating services and Composite services. These require very different migration procedures because of their different structures. Migrating services are Primary Service Cells, which are atomic units and cannot be divided to subcomponents. In contrast the Composite services, also called Service individuals, are composed of more than one service elements, which can be Primary Service Cells or another Service Individual.

In case of migrating Service Cells, the migration procedure contains the following basic steps:

- suspend the execution of migrating service entity
- migrate the runtime service data and states to a destination node
- initiate the new service with restored state on the destination
- resume application execution

In case of Service Individuals the migration procedure is more complex, because the migrating Service Individual may be composed of many Service Cells locating in different nodes. Therefore, there are a lot of possible combinations. For instance, it is possible to a) only migrate the graphs of a Service Individual, b) migrate also all the Service Cells constituting an instance of the Service Individual, c) migrate only some of these Service Cells constituting an instance. Additionally, the migration of multiple Service Cells raises the question whether to migrate them to one or several nodes. While the migration of Service Individuals is part of our ongoing work, we started with specifying the migration of Service Cells.

Each node (at least U-nodes) hosting BIONETS services contain control logic which is responsible for monitoring and managing the service environment. The logic is responsible for triggering the migration behaviour, for example, because of lack of resources in host node or changes in network connections. The migration could be also triggered based on user behaviour and/or context information. The node's platform provides the migration functionality or host a service which implements the functionalities necessary to support the migration of service to (nearby) node. The overhead associated with packing, transferring and rebuilding runtime services is relatively high, so the decision when and where the service needs to migrate has to be carefully considered.

There are many issues that need to be considered in the BIONETS system in order to achieve scalable and efficient procedures for service mobility; these future research directions for tasks in WP3.1 and WP3.2 include i.e.:

- Which forms of Migration are implemented? Code migration, i.e. the copying of a complete service (not only its description) to other nodes.
- Alternative migration procedures (in case code migration is not possible) as migrating service descriptions only or service provisioning where a node can request another node to compute only the result for a given problem instead of transferring a service that can compute the result?
- How is service migration avoided in case the receiving node is not able to cope with the requested service (e.g. because of spare resources on the receiving node)?
- *Swarm intelligence (SI):* Many self-organizing biological systems are based on the principle of swarm intelligence. In an SI system, intelligent behaviour emerges from the numerous interactions of simple subcomponents. One of the many mechanisms at work in natural swarms is the use of pheromone trails to select the shortest path between two locations. Ant colonies apply this principle, which has inspired a new research direction called ant colony optimization. E.g. [28]
- *Game Theory (GT):* Game theory deals with multi-entity decision making, in which each decision maker tries to maximize his own benefit. Game theory originates from economics, where it has been used for long in understanding various contexts, but it has been applied in various other fields. Today, it is applied to telecommunications as the users try to ensure the best possible quality of service. It has been used, e.g. for solving network problems such as congestion control [32] and cooperation among nodes in ad-hoc networking [33]. Autonomic service agents with varied interests characterize BIONETS service environment quite well. BIONETS services are not following some centrally mandated algorithm and they are acting mainly for their own interests, rather than that of the system. Game theory, which studies the interactions of multiple independent decision makers that try to fulfil their own objectives, appears to be a natural tool for both designing and analyzing the co-operation and interactions among such agents [34].

3.6.1.1 Service Migration Procedure

In practice if the service is implemented for example in Java, the migration could be transmission of the service class name, class code, and runtime data state. All the service data state is serialized at an origin node and de-serialized on a destination node by using the Java serialization mechanisms. A destination node's platform loads the received class definition into JVM, and then instantiates the service with the received state.

As depicted in Figure 3-5, the service migration mechanism proceeds as follows:

1. **Initiation:** When the Service Execution Framework (i.e. the node) detects that some resource in the underlying physical system is overloaded or to be unavailable, based on some performance data collected, it issues a "ToMigrate" message to a Service component, along with the addresses of the source and destination nodes. The "ToMigrate" could be also initiated based on the user actions and other context information, e.g. request from other node. Depending on the cause of the migration the migrating Service can either move itself or make a replica of itself and move the replica to the destination node.
2. **Packing and wrapping up**
3. **Moving**
4. **Restoring and re-configuration**

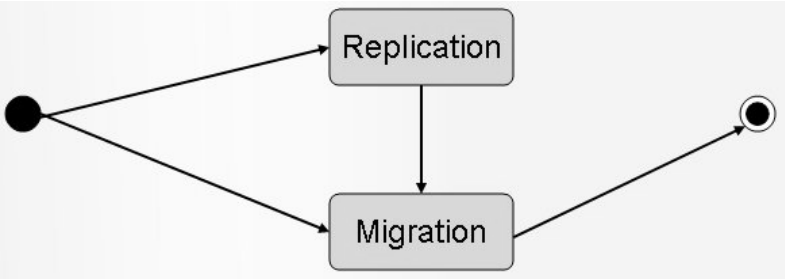

Figure 3-5: Phases of Autonomic service migration

States:

- Replication — some external or internal condition/event triggers the service to initiate the replication and migration of the service to another node. For example the service is too "popular" and cannot fulfil the entire request.
- Migration — some external or internal condition/event triggers the service to initiate the migration of the service to another node. For example the there is lack of resources in host node or the service needs to "follow the user" to a new device/node.

3.6.2 Vertical Service Deployment

In order to control the service life cycle the BIONETS service framework includes features that require from the current point of view partially centralized management components, integrated as Mediators in the service framework. The management adds a certain degree of processing overhead to the actual service functionality to make the services and the entire service framework behave autonomous. This overhead begins with the processing of specially encoded data exchange among service and nodes and may for instance end with cooperative processes like execution control or distributed service request processing. We argue that in the best case the execution of a Service Individual should not require more physical resources then needed by the actual functions behind the composing services in order to also address badly equipped devices as represented by T-Nodes. Thus, it is worthwhile to abstain from external entities (no matter if centralized or decentralized) controlling composite service execution if possible. To prepare the achievement of this goal the vertical deployment process is supposed to partially bypass the dedicated control components (i.e. the mediators) and configure all services composing a Service Individual in such a way that service output data are directly send to the respective next services in the composition graph. Additionally the submitted data are preformatted in a manner that allows the receiving service to directly

interpret them. That way a cascading service network is established, possibly spanning services spread across different nodes.

Although, there is in general no guarantee for lasting (direct or indirect) connections between two arbitrary nodes in the network and no continuous access to superior, reliable network infrastructure serving as backbone or fallback, sets of nodes temporally form islands (e.g. while being physically close to each other). For the vertical deployment and the aimed results these islands are expected to be long-living. Possible example can be found in immobile office or home environments, which are open but relatively stable in their composition of devices and services respectively. The communication inside these islands is less constrained, i.e., for certain time spans there is an exceptional high probability for reliable interactions among nodes, but nonetheless no guarantee at all. However, as Service Individuals are only blueprints, the vertical deployment starts with finding an appropriate binding for the Service Individual, i.e., substitutes for the service placeholders in the composition graph. Subsequently, the services composing a Service Individual are set up to execute the logic of the Service Individual self-reliantly, e.g., by starting execution on certain events and publishing trigger events themselves. For those cascading executions the service bindings need to be available in a long-term fashion but the consumption of resource is supposed to be significantly confined.

3.7 Service Execution

The central idea of service-oriented computing is to decouple service provider and service consumer, while allowing the service consumer to dynamically bind a required function as, e.g., claimed by Garlan [Garlan04]. Instead of defining fixed dependencies to (remote) components when implementing an application, only the type of required functionality is determined in the code and an appropriate implementation is searched and utilized at runtime. For example a call setup service can be instantiated by a mobile phone, a PSTN phone or a VoIP softphone. That way, the utilization of a service can be adapted to runtime condition, as, e.g., availability, quality or even price.

In the BIONETS Service Framework we do not claim that each service request can be answered by a single Service Cell, instead Service Individuals, i.e. compositions of multiple Service Cells are supposed to realize most of the required functionality. The Service Cells required when instantiating/binding a Service Individual are spread across the network. For that reason service provision requires remote interactions between the nodes of the network to execute composed services. These interactions are constrained by the characteristics of the network. For instance they need to cope with varying delivery delays, intermittent connections and even disappearing nodes.

A delicate question regarding composite services is: what are their semantics in the face of partial failure. If a composite service is described as an aggregation of the results of two existing services, and the execution of one of them fails, does the whole execution fail? Is the whole composite service executed again? Depending on whether the execution of the existing services has side effects, this may be feasible or not. For example, if one of those services implies making a payment, and the execution of the payment service fails, has the payment taken place already? At which point did it fail? Should the composition try to run the service again and risk a double payment? Should the payment service provide atomic transaction semantics? If both services have side effects and one of them fails, should the other one be rolled back? Can it be rolled back?

Ideally, these questions should be answered by appropriate data in the service announcement, so the composition may take a good decision. However, the applicability of the BIONETS needs to hold for the identified scenarios, but none of the scenarios described in deliverable 3.1.1 seems to need atomic transaction semantics. For that reason we will consider transactions in later iterations again and focus on the management of the communications in unstable ensembles of peers for the beginning.

However, the execution of services can be divided into two different classes. The first, and most basic, is the "native" execution of service components. This class represents the atomic services provided by peers in the BIONETS network, the Service Cells. The second class is based on the cooperative interpretation of the abstract Service Individual description. It basically refers to workflow-like services that are composed of other Services. The component services need not be native services. For the execution of the first class of services, the native ones, the BIONETS only need to route the requests into the service endpoint, and route the responses back to the caller.

The execution of the second class of services is performed by the BIONETS platform itself. These services provide an abstract description of the various component services that are to be executed, and the relationships between them in the form of inputs, outputs, and decision points. At least, a Turing-complete language would be used to specify the composition of other services, and the specification of the composition is executed by the BIONETS platform.

While controlling the execution of one Service Cell is a matter of the hosting node, controlling the execution of a Service Individual composed of several Service Cells and other Service Individuals is much more complex. At moment we have identified three possible approaches to address this challenge, 1) centralized control, 2) decentralized control, 3) embedded control.

3.7.1 Centralized Control of Service Individual Execution

In an approach that utilizes centralized control, for instance one U-Node of the environment could exclusively perform the control of the execution process. This controller could take the functional graph of the Service Individual, discover appropriate Services matching the nodes in this graph and execute these services one by one until the functional graph is completely processed. Additionally it supplies the single services with input and obtains the outputted data for further use. When a service fails the controller discovers an appropriate alternative and recovers the execution at the point it was interrupted. Although the approach of a central execution controller promises a good maintainability and an acceptable complexity of implementation, it does not cope with the expected dynamic of the environment. For example, if the execution controlling node is physically separated from the ensemble of nodes providing the required Services, a possibly already started execution process breaks down and needs to be started again under the control of another U-Node.

3.7.2 Decentralized Control of Service Individual Execution

Due to the dynamic nature of the network, the corresponding fluctuation in the ensemble of nodes and the choice of services, a cooperative control of the Service Individual execution seems more appropriate. Dedicated control entities on all U-Nodes of the service environment should be aware of ongoing service execution processes and able to attend them when required. For this purpose we envision a self-determined kind of service execution. All U-Nodes of the service environment are notified about the required execution of a Service Individual. Accompanied by the notification the execution graph is distributed among the U-Nodes. Subsequently, each U-Node decides on which part of the graph to execute. The decision is supposed to be based on the set of hosted services and the current situation of the nodes (e.g. physical capabilities, owner preferences, etc.). The interactions of nodes are supposed be monitored by other nodes involved in the execution process. For this purpose we envision the utilization an appropriate interaction model. A possible candidate, the Semantic Data Space, is briefly introduced in section 4.2.4. However, the awareness of ongoing interactions allows the recovery of interrupted or failed service executions and inherently benefits the reliability of the service environment. An example is depicted in Figure 3-6.

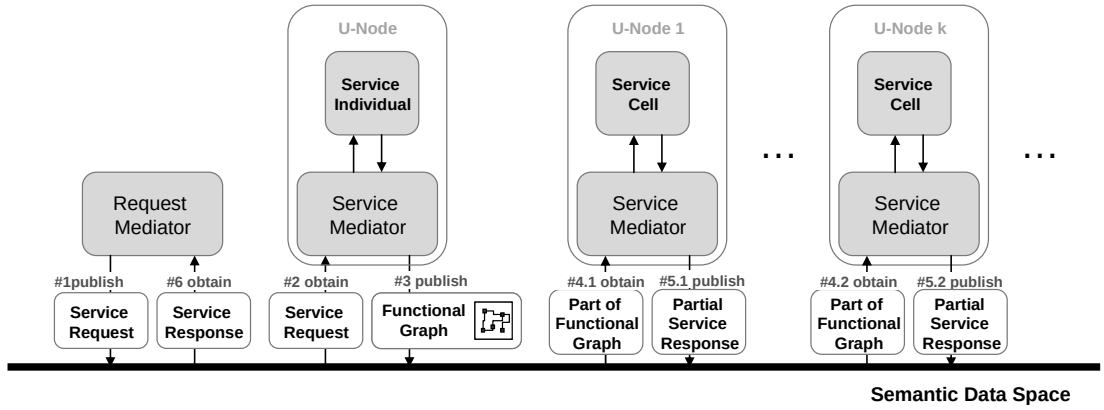

Figure 3-6: Service individual execution under decentralized control

3.7.3 Service Individual Execution without External Control

The third approach of controlling the execution of Service Individuals cannot be directly compared to the other ones, as it is based on particular assumptions about the nature of the BIONETS environment. As already outlined in section 3.6.2 the BIONETS environment, at least ensembles of nodes are supposed to be partially stable in their composition. Those stable ensembles are also referred to as islands of nodes. When the dynamic of the environment can be ignored, the need for a co-dependent control of the service execution by separate entities is also reduced. Hence, instead of having an external controller that triggers the execution of a Service Cell, provides inputs and obtains outputs, this approach aims that services pass there outputted data and the execution focus themselves on to the respective next service in the functional graph of the Service Individual. For this purpose Service Cells need to be configured in an appropriate way, a process which is addressed by the Vertical Service Deployment. Consequently, the execution of a Service Individual is - once started - comparable to a non-interrupted cascade of Service Cell executions.

3.8 Service Deprecation

The main goal of service deprecation mechanisms in BIONETS is to save resources (mainly storage) by no longer maintaining SCs which are no longer in use, either because they offer an uninteresting service or because they are outperformed by other competing SCs. Service deprecation in BIONETS can be seen as an outcome of the natural selection process, i.e., the disappearing of services not able to achieve and maintain an adequate level of fitness. Also, ageing is something to take into consideration in BIONETS environment. Indeed, BIONETS - as all evolutionary systems - tends to suffer from a problem which is known as the "rich gets richer" paradigm [71][68]. These mechanisms tend inherently to favor "old" organisms (i.e., SCs) with respect to new ones. In this case, an opportune mechanism has to be implemented to slowly decrease the fitness of a service over time, so to let space for newcomers to grow and evolve.

Since, for the moment, autonomy is integrated in BIONETS at the node level, this can be used to implement service deprecation as well. This consists of two mechanisms:

- *service ageing*: the fitness of a service is decreased over time according to a function *g(t)*, where *t* represents the age of the service, i.e., the time elapsed from the first moment it has been loaded on the node. The function satisfies *g(0)=1* and tends to *0* as *t* tends to *infinity*. Clearly, the timescale over which the function *g* decays has to be selected in such a way not to interfere with the service execution process. Roughly speaking, the service ageing is a very “slow” mechanism, which induces (through the natural selection process) a sort of garbage collection task.
- *service erasing*: once the fitness of a locally maintained SC falls below a given

threshold, the SC is eliminated from the system to free storage resources.

Both the tasks detailed above can be implemented in a totally decentralized way, since they are based on local decisions (taken inside a node) on local copies of a service only. In principle, it would be possible to also envision mechanisms for "actively" deprecating a SC, so that, for example, a SI can force the erasure of a given malfunctioning/malicious SC. However, it is clear that such mechanisms pose a lot of issues in terms of security and trust, so that we decided, for the moment, to exclude them from the picture.

4. Service Interaction Models

BIONETS addresses service architectures in distributed, highly dynamic systems, characterized by the mobility of network nodes and intermittent connection due to wireless links. The service architecture aims at interworking in loosely coupled ensembles of entities spread across several nodes (Service Cells, Mediators, any other active components etc.). In the simplest case this interworking is reflected by scenario wherein one entity requests a service, i.e., usually the execution of a task, while another one fulfils this task and responds with the results if appropriate. Regarding data exchange among the entities in the environment from a more application and services dependent perspective leads us to the notion of interaction. Interactions reflect the subject of communication and data exchange. As implied by service framework and service life-cycle, there are various processes that require cooperation among multiple 'autonomous' entities. An example can be found in the execution of an SI, whose composing SCs are spread across several nodes of the systems.

For this purpose, in deliverable D3.1.1 [81] we have introduced an intermediate layer between the BIONETS networking support and the actual service architecture, referred to as interaction framework. The interaction framework serves as base for the introduction of different interaction models to the BIONETS Service Architecture. In this context an interaction model defines a way several parties connected to the same communication medium may interact with each other. Since each middleware is based on a certain interaction model the interaction framework may also be seen as placeholder for different middleware concepts. Some examples for different interaction models are realized in message based systems, publish-subscribe systems, remote procedure call, or tuple spaces. An interaction model does not define the purpose of interaction, i.e. it is defined independent from the actual utilization. Common characteristics determined by an interaction model are listed below.

- Primitives for the exchange of information, e.g. send(message, destination) and listen() for message based systems
- Roles for the interacting parties, e.g. publisher and subscriber in publish-subscribe systems
- Representation of data to be exchanged by the interaction model, e.g. tuples, chunks of plain text, hierarchically typed events

The interaction framework is based on the notion of resource manipulation. To keep the framework simple resource manipulation is realized through the least set of required operations, namely Create, Retrieve, Update, Delete; an approach that showed to be very promising as argued in by fielding in [92]. Consequently, interaction models are supposed to be implemented through resource manipulation. The interaction framework provides a simple interface to be mapped to the actual network layer, i.e. in our case the BIONETS networking support. The shortcomings of the network (e.g. intermittent connections, disappearance of nodes) are neither transparent nor abstracted, but rather inherited upwards and therefore to be addressed by the interaction model or even the architecture above. An advantage of this approach is the easy exchangeability of implementations for interaction models without touching the lower or higher layers. Moreover, interaction models can be varied on demand. That way, interaction models can be subject of evolution themselves.

4.1 Interaction Framework

The interaction framework is build up on a set of few atomic operations, namely *Create*, *Retrieve*, *Update*, and *Delete* (CRUD). The four operations expect a return path to inform the caller of the status of the operation. In particular, Retrieve has absolutely no meaning if no such path exists. The other three may be performed without the return path, though its lack severely restricts the reliability of the system. However, it is not said that the response has to be received immediately.

The disappearing network paradigm severely restricts the modes of operation of the services. This is because return paths for requests of any type can only be guaranteed for very short time scales. In longer time scales, the client the response is being sent to may be out of reach of the service that wants to give the response back. The network layer *must* provide the return paths for the CRUD interface to be meaningful. But, also, the network layer must provide the return paths for any kind of interaction. Otherwise, it's not interaction, but {uni,multi}casting.

It should be possible to follow the CRUD interface to provide all kinds of services. For example, an announcement for a service would be a *Create* on the network. A change on an announcement would be an *Update*. An operation which only gets data and has no side effects should be a *Retrieve* operation. A cancellation of an announcement or request would be a *Delete*. This is a departure from the traditional RPC-style of systems interaction. However, it is hardly new, as the Structure Query Language (standardized ANSI in 1986, but created in the 1970s) [\ref] follows the CRUD interaction style.

The main difference is that while the RPC-style focuses in procedures as the main interaction mechanism, a CRUD interface focuses in the manipulations of data resources. RPCs (and procedure calls in general) usually provide an interface of many different procedures with different names and/or parameters to perform various operations. A CRUD interface provides a small, standardized set of methods that are applied to a larger set of resources.

To make the idea clearer, an example could be used. Assume there is a database of users, and a new user is to be added. Following an RPC-style, a new procedure called *new_user* would be defined. It would accept the user id as a minimal set of parameters, and probably the real user name and some other information as well. In SQL, this would mean an INSERT INTO statement would be executed. Notice the procedure used, INSERT, is the same for adding new users as for any other type of data addition.

The important point is that in RPC the semantics of the operation are opaque, and the underlying platform must support RPCs in a transparent way. In the CRUD style the semantics are well defined. The underlying platform may support the operations in a transparent way, or provide facilities. The fact that the underlying platform is able to provide facilities for the execution of the remote actions is important. The semantics of the RPC *new_user* and of the SQL INSERT INTO statement are the same: add a new user. However, in the RPC case the underlying platform does not know what the operation does. Is it a query? Is it an update? Is it a request for deletion?

The main difference between the RPC style and the CRUD style lies in the fact that the RPC style does not define the semantics of the various methods, while in the CRUD style the semantics are well-defined. This is possible because the number of procedures/methods/verbs in the CRUD style is small, and can hence be standardized, while in the RPC case the number of methods is arbitrary and left to the mercy of the system developers.

The standardization of semantics and the support provided by the underlying platform are important points when the remote interactions belong to long time scales. In the networking domain, error detection is performed primarily by the means of reckoning the passing of time. If a request is sent and no response is received in a bounded amount of time, the request is assumed to be lost.

While this is not a big problem for retrieve-only operations (for example, pointing a web browser to a site and receiving the time-out error), it is a big problem for operations that change the state of the resource they are applied on. For example, in a banking scenario, money withdrawal has to be performed exactly once. If the request for withdrawal is lost, the requester (the user) doesn't get the money. If the response is lost, the user doesn't get the money, but the amount has already been subtracted from his account.

To avoid this situation, flow control techniques may be used. But as the platform providing the execution environment for the RPC is unaware of this situation, the burden is on the RPC layer itself. The simple *withdraw_money* procedure starts to multiply into a set of confirmation and acknowledgement requests and responses that have nothing to do with the original, simple procedure. The CRUD interface is not immune to this. But as the semantics

of the (few) procedures are well defined, the underlying platform may (and should) provide the means to ensure the exactly-once semantics in a way that does not encumber the original procedure (*Update* in this case). It does not encumber it because the interface already provides this confirmation-acknowledge message set. More important than this, three of those four operations are idempotent. That means they can be repeated as many times as needed by the underlying platform, and the effects of two or more repetitions are the same as the effects of a single request. Creating a resource is an idempotent operation. If a new user was to be created by using the *Create* operation of the CRUD interface, all the needed information would be provided when the operation is performed. If the operation was repeated, the exact same resource would be created again. Thus, the resource is created independently from the number of repetitions. Deleting a resource is also idempotent. The results of the first successful request are that the resource is deleted. The results of the subsequent successful requests are an indication that the resource no longer exists. After any number >0 of requests for deletion of the resource, the result is the same. Obtaining a resource, of course, is idempotent. The resource does not change as a result of the retrieval operations. It may change between retrieval requests as a result of update operations performed by other entities in the system, but not as a result of the retrieval operation itself.

The update operation, however, can not be repeated at will. An example of an update would be *"Withdraw 100€ from my account"*. It is evident that the result of one request is different than the result of five of those requests. Therefore, its use should be minimized. The adoption of the CRUD interaction style represents a change from the traditional view of remote procedures. It implies a change in the way remote applications are perceived, designed and implemented.

4.2 Loosely-coupled Interaction

The interaction model specifies how data is exchanged among several participants of an interaction. Therefore, it abstractly describes an interaction medium and the ways participants communicate via this medium. Regarding dynamic systems, as given in the context of BIONETS, leads us to considerations about interaction models and their immanent constraints, especially to the strength of coupling between the interacting parties. For instance Remote Procedure Call (RPC) is not an adequate interaction model as the callee cannot be expected to respond in a negligible time span. In [41] Cabri et al. classify interaction models according to the strength of coupling between the interacting parties. Usually, time and space are regarded as most significant characteristics, but due to the heterogeneous setting of the computing environment, we will also consider the representation of data as driver for loosely coupled interactions.

4.2.1 Coupling in Time

Temporal coupling in general requires all parties involved in an interaction to be synchronized in time. Hence, the time is neglected and delays in data exchange are supposed to be minimal. In contrast, decoupling in time explicitly disregards temporal dependencies, i.e., delays in data exchanges are expected.

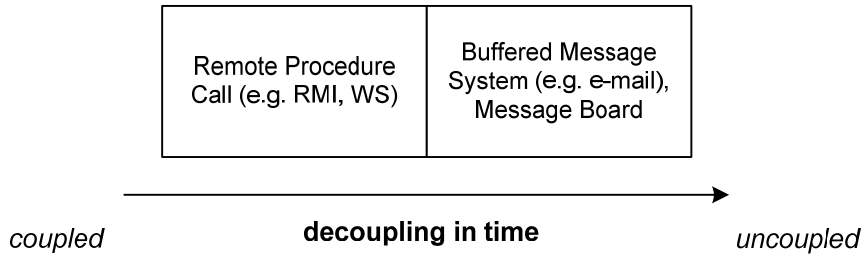

Figure 4-1: Classification of interaction models with regard to temporal coupling

A common example for a temporally coupled interaction is the Remote Procedure Call (RPC). Here one party performs a procedure that was previously requested by another one.

Until the procedure is finished the caller is usually blocked while awaiting the results. RPC is especially applied in middleware that is intended to support the interoperability of heterogeneous systems while hiding the distribution. Technologies like Java Remote Method Invocation (RMI) [42] and Web Services are modern relatives of RPC.

For instance buffered message systems or message boards are representatives for temporally uncoupled interaction models. The first one includes an ordered message storage that is temporally coupled with the interacting parties and consequently prevents the parties from being directly coupled in time. Implementations of those systems are the Java Message Service (JMS) [43] and the common e-mail system. Message boards are similar to buffered message systems, but they do not provide a fixed order for the stored messages. Moreover, in contrast to common message systems they require the addressee to poll a message instead of delegating it automatically.

4.2.2 Coupling in Space

Spatial coupling implies that the parties involved in an interaction know each other, at least by name. In contrast, a spatially decoupled interaction allows the parties to stay anonymous. They are neither supposed to know each other, nor how many other parties are involved in an interaction. Therefore, interactions are usually based on the contents of the data exchanged through the interaction medium. Spatially decoupled interactions are especially favorable in the BIONETS computing environment and its archipelago-like, disconnected nature. In BIONETS we cannot rely on the addressability of functional resources (e.g. service and mediators). Moreover, coordinative tasks may need to support one-to-many communications.

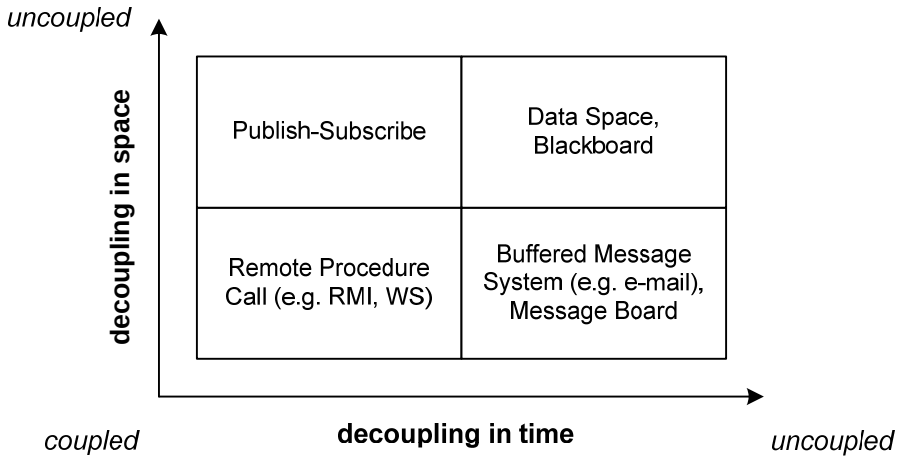

Figure 4-2: Classification of interaction models with regard to temporal and spatial coupling

Spatially coupled systems are, e.g., message systems, wherein sender and receivers are addressed directly in the message. Examples are again the common e-mail system, or JMS. The most appropriate representatives for the realization of spatially decoupled interaction models are, e.g., publish-subscribe approaches, data spaces and blackboard systems.

Publish-subscribe approaches are anonymous messaging systems. They usually support two interaction directives, publish and subscribe. The subscribe-directive allows parties to sign in for the receipt of data from the interaction medium. While subscribing, the parties also define a filter criteria for messages they are interested in and a callback address for the notification about the occurrence of an appropriate message. Due to the undirected nature of message delivery, messages are rather referred to as events. Emitting a new event is done with the publish-directive. Consequently, the interaction medium is supposed to verify the filter criteria of the subscriptions on all new events and to delegate the events to the corresponding subscribers. Eugster et al. identify in [44] three major methods for event filtering, by topic, by type and by content. Type- and topic-based filter criteria depend on the events to be either typed in terms of programming languages or tagged with a key word, the topic. Both, types

and topics, may be organized hierarchically. Hence, the filter criteria are defined to match particular types or topics, or appropriate super-types and –topics. In contrast, content based filter criteria refer to the data composing an event and may, for instance, be represented by conditional expressions or program code.

Publish-subscribe approaches can either be realized in temporally coupled or a temporally decoupled manner. Temporally coupled realizations require all subscribers to be available at the moment an event is published. Delegations of events to unavailable parties are discarded. Temporally loosely coupled approaches queue events for subscribers until they are available again.

Data spaces in terms of Gelernter et al. [44] represent a spatially and temporally decoupled interaction model. They are similar to message boards, but the contained data are not individually dedicated to certain parties. The parties are rather interacting while reading from the data space, adding new data, and manipulating existing data. Accordingly, the original interactions directives are read, in, and out, whereof out allows removing and reading data from the data space in a single step. However, data are retrieved in an associative manner like in publish-subscribe approaches, i.e., data units within the space are addressed by templates or queries. If there are multiple data units matching a template, one is chosen in a non-deterministic manner.

Recent implementations of data spaces like MARS [46], JavaSpaces [47] and EventHeap [48] include publish-subscribe functionality in addition to active interaction directives. Hence, a party is no longer required to continuously poll the data space to recognize changes within the contained data. Instead, templates can be subscribed and newly inserted data is published immediately. Nevertheless a major difference between data spaces and publish-subscribe systems is the management of states. While temporal decoupled publish-subscribe systems queue the data related to subscribers, in data spaces there is no distinction with regard to the possession of data. Thus, as long as data are kept within the data space, they are accessible for all parties.

An interaction model that is consciously aligned to the creation of a common knowledge base is implemented by blackboard systems in terms of Erman et al. [49]. Blackboards are shared storages, similar to data spaces. Since blackboards are designed for collaborative problem solving, the parties compile several solutions by alternately adding facts, i.e., data with a particular meaning to the blackboard. Blackboard systems are based on publish-subscribe approaches. Hence, parties can subscribe for changes within the knowledge base and may manipulate this knowledge base when being notified. Concurrent manipulations are managed by a controller. The controller determines the appropriate next participant that is allowed to write to the blackboard on every change. During this process each party can view the whole knowledge base. Therefore, on the one hand the parties are required to interpret the blackboard data on each change they are notified about. On the other hand they need to understand the whole problem domain or at least a reasonable part of it. However, in contrast to data spaces blackboard systems rather represent data units that explicitly refer to each other than discrete data unit without any relations.

4.2.3 Coupling in Representation

Especially in spatial decoupled systems the contents of exchanged data are the significant driver for interaction. Although the participants are not required to share a common namespace in terms of addresses, they usually need to agree on common data structures, vocabularies and vocabulary semantics to describe the data communicated among them. This holds for the nodes and mediators as well as for the services. As the BIONETS service environment is supposed to be heterogeneous and self-evolving, it is to rely on a standardization of terminology. Instead, we envision to loosen the coupling in representation. The intention of loose coupling in representation is to allow the parties involved in an interaction to only share minimal knowledge about an application domain and the terminology describing this domain. Therefore, the language describing data during an

interaction needs to be ad hoc interpretable, while the interacting parties should be able to understand the language and conceive the semantic. Thus, decoupling in representation addresses on the one hand the representation of data and on the other hand the representation of meaning. In the following we will concentrate on the representation of data.

A data structure defines a formal order for an atomic set of data units. Generally a structuring of data benefits exchange and processing of information through machines. While free text documents are regarded as unstructured, tuples, tables or trees are simple examples of structured data. More comprehensive representations of data structures and the dependencies between several segments of the structure are provided by relational and object relational data models, originally introduced by Codd et al. [49]. These models are especially employed for large amounts of co-related data in data bases. The appearance of such a comprehensive structure can be described by a schema.

However, while interacting in an open, distributed environment as the BIONETS system, the interacting parties are required to either implicitly share a common schema or publish the one they will utilize. That way other participants are enabled to process the corresponding data. Publishing means to refer to an open accessible schema or embed the schema within the submitted data. Those approaches of self-describing data are referred to as semi-structured like argued by Abiteboul et. al. in [51]. Today's most common format for semi-structured data is the Extensible Markup Language (XML) [52]. The XML standard defines how simple datasets can be structured and composed to trees. Vocabulary definitions and structural constraints on data, e.g., typing or the number of child-nodes per node, can be specified with XML Schema [53] for each document type. Additionally, XML Schema allows the combination of terminologies and structures from different document types in a single XML document. Therefore, any known scheme may be referenced and included, while namespaces support the differentiation of vocabularies from different schemes and e.g., the distinction of terms that are defined twice.

XML is designed to describe data structures in self-contained documents. Consequently, there is no common method to represent relations among several documents and their contained data. If, for example, two parties in a temporal decoupled interaction exchange XML documents that are required to refer to each other because of representing a dialog, this reference needs to be represented in proprietary manner. For that reason in addition to XML the Resource Description Framework (RDF) [54] was introduced. In RDF, relations between resources are represented by triples that contain a subject, a predicate and optionally an object. Subject and object identify the related resources, while the predicate determines the kind of relation between them. The resources are either defined in place or referenced with Uniform Resource Identifiers (URI) [55]. Objects can also represent atomic data values (e.g. floating point numbers and character strings) or may even be left blank. Combining triples results in a directed graph that spans, if desired, the contents of several documents. An example graph is illustrated in Figure 4-3, wherein resources and predicates are represented by simple names instead of complete URIs. However, for instance the name Bob could also represent a URI like 'http://www.example.net/names#Bob'.

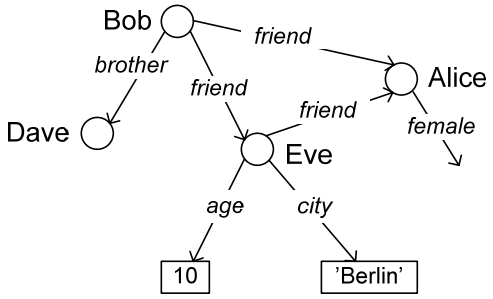

Figure 4-3: Simplified illustration of a graph in terms of RDF

4.2.4 Semantic Data Space

Loosely coupled interaction models weaken temporal and spatial dependencies, while requiring only minimal correspondence of data representations supported by the interacting parties. In general, Data Spaces provide a suitable combination temporal and spatial decoupling, while to interweavement of the loose coupling in representation as further aspect of interactions led us the concept of the Semantic Data Space (SDS). SDS serves as one interaction model for the BIONETS service architecture that especially addresses loosely coupled interactions. For that purpose SDS addresses the required features identified above to enable loosely coupled interactions among multiple parties in the BIONETS computing environment, while basing on state-of-the-art technologies as a start.

4.2.4.1 Organizational Structure

The description of the organizational structure of the SDS comprises considerations about the interacting parties, data representation, and the organization of the data storage. For the moment Mediators (e.g. Request Mediator, or Service Mediator) are regarded as interacting parties. Mediators interact indirectly by writing data to Data Spaces and reading data from those Data Spaces. SDS composes multiple Data Spaces to one coherent environment, the so called Data Space Environment. Therein, each Data Space is supposed to contain only data that concern the same subject. Accordingly, Mediators are allowed to control the Data Space Environment to introduce new subjects that require interactions and consequently new Data Spaces.

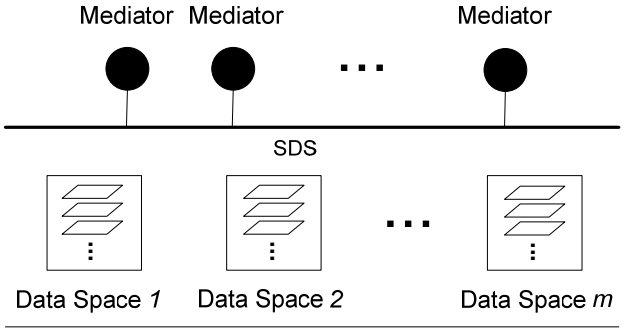

Figure 4-4: Abstract structure of SDS interaction model

4.2.4.2 The Data Space Environment

Interactions are not necessarily limited to a two-way information exchange between two parties. Instead, they are often characterized by the exchange of various data about a particular interaction subject, wherein these data are originated from multiple parties. Moreover, interaction may take very long, even hours or days. These durations can be caused by the processes that correspond to the interaction subject. If such a subject for example addresses the playback of a movie, the interaction may first be regarded as finished if the playback is over.

For these reasons the organization of the entire interaction model is simplified by storing data exchanged during interactions with regard to the addressed subjects to different Data Spaces. Consequently, one Data Space always reflects all communicated data concerning the same interaction subject.

A Data Space needs to be opened by a Mediator that originates a new interaction subject and accordingly some data describing this subject. The same Mediator that opened a data space is also intended to close this space afterwards. In addition to the explicit removal of a Data Space, each Data Space may be given a finite life time by defining the maximal time that is allowed to elapse between two interactions. If the time is exceeded, the Data Space is intended to be closed automatically. However, all open Data Spaces are accessible to all Mediators in the scope of the Data Space Environment.

4.2.4.3 Data Spaces

The Data Space is the actual virtual storage device for data exchanged among the participants and consequently reflects all data concerning a particular interaction subject. In section 4.2.3 the need for a non-proprietary representation of relations between data that originated from different parties is identified. Those relations are required to reproduce the context of data exchanged during an interaction. If for instance two parties alternately write data to a Data Space that refer to each other in a directed manner, a third party is allowed to obtain the developing of the interaction from the references. This characteristic is especially relevant if the third party only recognizes an interaction when data were already exchanged, i.e., the party is not aware of the chronological order of the data.

For that reason the data inside a Data Space are represented as directed graphs in terms of Resource Description Framework (RDF) [54]. Each node in such a graph is given a unique name and a set of attributes. Moreover, a node forms a set of relations to other nodes. Attributes may be regarded as relations to data nodes that represent constant values like integers, strings or floating point numbers. Relations are the edges of the graph and are also given a unique name. In contrast to the nodes, relations may occur multiple times in the same graph while connecting different pairs of nodes.

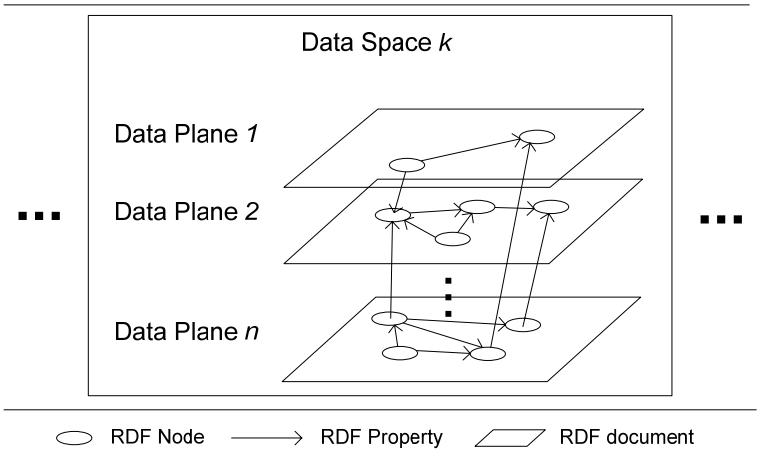

Figure 4-5: Visualization of one data space with multiple data planes

A graph is allowed to be dynamically extended with further groups of nodes and relations. An extension is rejected by the Data Space if such a partial graph is inconsistent to the one it should extend. Inconsistency may be caused for instance by duplicate node names or any violations of structural constraints on the defined graph semantics. However, to separate the spatial growth of a graph from the temporal growth, the Data Space is divided in multiple Data Planes. Each new partial graph added to the Data Space is stored to a separate Data Plane, while the data of all Data Planes together represent the entire graph of a Data Space. Therefore, relations may also address nodes from different planes as illustrated in Figure 4-5. The separation of the Data Space to multiple Data Planes allows reproducing the history of an interaction independent from the data semantics. That way, rollbacks on data can be performed that result in consistent states.

However, using strongly related graphs for the representation of data also affects the removal and consequently the change of data. For instance removing a single Data Plane that contains nodes related to nodes of other Data Planes may result in inconsistencies. For that reason the removal of data from Data Spaces, similar to the withdrawal of tuples from tuple spaces, is not allowed. Since each Data Space is intended to contain just data addressing a particular interaction subject, the entire Data Space is rather supposed to be closed if there are no further interactions concerning the corresponding subject. To substitute the synchronizing effects realized by withdrawing data, for instance new partial graphs may be added to the Data Space that mark fragments of the entire graph as locked. Therefore, all interacting parties are naturally required to know that a special data structure represents a lock. The exclusive access is then only enforced on a semantic level, but the concurrent retrieval of data is still possible.

All information needed to reproduce the context of interactions for a particular subject can still be obtained from the Data Space.

4.2.4.4 Patterns

With respect to the original tuple space concept, reading data from a Data Space is also realized in an associative manner. For that reason data are retrieved by matching patterns on the contained graph. The patterns themselves are graph structures consisting of nodes and relations. However, not all nodes and relations in a pattern are required to be named explicitly. Instead nodes may be represented by free variables that serve as placeholders in the pattern graph. If any sub-graph in the Data Space matches the pattern i.e., a graph can be found that contains substitutions for the free variables while holding the structural constraints of the pattern, this sub-graph is regarded as pattern binding. While testing patterns in a Data Space the entire graph spanning all Data Planes is regarded. An example for a graph pattern is illustrated in Figure 4-6.

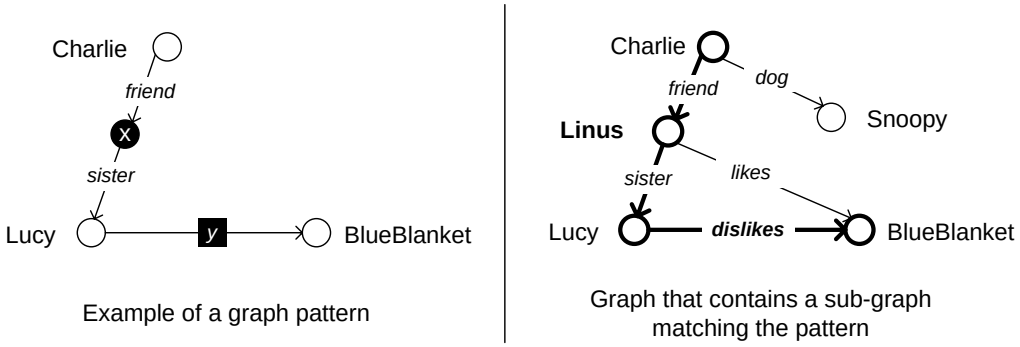

Figure 4-6: Visualization of a simple graph pattern

4.2.4.5 Data Manipulation

In addition to the organizational structures the following use cases of the Mediators determine the interaction directives of SDS. Generally, the interacting parties are permanently allowed to read data from the Data Spaces and add new data. Only the removal of data is not supported to avoid data inconsistencies as explained in 4.2.4.3. The blocking retrieval of data provided by the original tuple space operations is substituted by a reactive retrieval as in JavaSpaces [47] or EventHeap [48]. Instead of waiting for some data to appear the parties interacting via a Data Space subscribe for these data while registering a graph pattern and a callback that is triggered on the appearance of data matching that pattern. Since all data exchanged about a particular interaction subject are kept in the Data Space, a subscribed pattern may also be evaluated on data that were added to the Data Space before the subscription was registered.

As claimed earlier each interaction subject should be treated in an own Data Space. For that purpose the interacting parties may open Data Spaces and finally close them when no further interactions for a certain subject are required. However, a comprehensive view on the entire ensemble of Data Spaces in the environment allows finding Data Spaces that concern particular interaction subjects. This view supports reading data from the environment while testing one pattern against all Data Spaces at once. The retrieval is again allowed to be either active or reactive. However, the explained use cases for the Data Space Environment are summarized below.

- Open a Data Space
- Close a Data Space
- Add data to a Data Space
- Read data from a particular Data Space

- Read data from overall SDS (i.e. entire Data Space Environment)
- Subscribe for new data on a particular Data Space
- Subscribe for new data on any Data Space in overall SDS

4.3 Service Capability Acquisition

4.3.1 Directories

In traditional architectures like Jini(tm) and FADA, a user registers a service in a Directory, creating a centralized point of knowledge about services. In BIONETS, where nodes can appear and disappear rapidly, we cannot rely on the existence of centralized nodes, and a different architecture has to be devised.

4.3.2 Service Announcement

For a service to be known in BIONETS, its provider has to announce it. Announcement is performed right after deployment by a service provider. When a node announces a service, all neighbours will remember it for a given period of time, creating in this way a network of references, much more common in social environments. In this kind of network the existence of a service is known thanks to the referrers, which also can provide other information about the service, such as its quality or its level of trust.

Since the topology of a BIONET is highly dynamic, memory of nodes has to be very short. This is due to the fact that, as nodes appear and disappear frequently, relations among them are not long and stored knowledge about other nodes loses validity over time. Memory about neighbours has to be enforced by some mechanism such as an expiration system that deprecates services that do not exist anymore. One method that can be used is providing leases that have to be renewed periodically by providers in order to keep their services in its neighbours memory, and thus visible in the network.

When new nodes enter the network, they send a join message to all their neighbours that make them respond with the services offered.

4.3.3 Service Matching

Service matching is initiated when a Service Request is received by the Service Mediator. A request defines a query by a user: it specifies the parameters and constraints that have to be accomplished in order for a request to be fulfilled, and it also carries a context with attributes from the user that can be used in the matching or in order to choose between possible services.

Based on this information, the Service Manager determines which combination of Service Cells is the most appropriate to handle the request, in case it can be done in the receiving node, or if there is the need to send a part of the request to other nodes. Service Cells provide the building blocks of the service architecture.

There are these major cases:

1. One of the Service Cells in the node can answer the request
2. The request must be answered by more than one Service Cell in the node
3. No subparts of the request can be handled by this node's Service Cells
4. Some parts of the request can be handled by this node's Service Cells

Depending on the case, different actions have to be performed:

1. The Service Mediator passes the request to the Service Cell, which in turn returns a response (Match)
2. The Service Mediator decomposes the request and passes each sub-request to the corresponding Service Cell. Then it receives all the partial responses and constructs a response to be sent back to the originator of the request (Match)
3. The response cannot be handled by this node (No Match)
4. The Service Mediator marks the subparts of the request that can be handled in this node and passes it to another Service Mediator. (Partial Match). This Service Mediator repeats the steps and when all the subparts are marked they are resolved by the nodes that marked them.

4.4 Context Provisioning

In the BIONETS project we regard autonomy not only as the capability of nodes to adapt their behavior according to changes in the computing environment, but we also address the capability of services to adapt their behavior according to changes of user needs. Those changes may for instance mean that a user poses a new request to the system or simply leaves a room and does not need the services consumed in this room any longer. The conditions and characteristics of the entire environment, i.e. its constituting entities is usually referred to as context.

In this regard, the term ”context-awareness” denotes the ability of making use of the entity’s context [57]. Before being able to interpret the context of an entity, the context has to be represented in a proper format. Strang and Linnhoff-Popien [59] evaluated key-value, markup scheme, graphical, object oriented, logic based, and ontology based models with respect to several criteria, e.g., distributed composition and the level of formality. They concluded that the ontology based approach meets the requirements best and is therefore the most promising way of modelling context.

An ontology, as far as information technology is concerned, is an explicit formal specification of objects or concepts and their relationships. One of the main benefits of ontologies is inter-operability when using a uniform way of representation. Furthermore, implicit knowledge residing in ontologies enables reasoning techniques. Alongside the stated information, with ontologies it is possible to add significance to the plain facts allowing its verification and the inference of additional statements.

The context of an entity comprises heterogeneous information that is gathered from many different resources. Context-awareness presumes the comprehension and interpretation of the represented context. Ontologies meet these requirements perfectly and are therefore an adequate way of modelling context. Furthermore, they are flexible enough to adapt to the ongoing evolution in information technology.

In the scope of the BIONETS project the utilization of context information is supposed to benefit especially the creation and transformation of services. Context information represents a reference set of data that contains major characteristics of the user, his or her general needs and preferences and current physical environment. In the scope of a human-centric service environment this information is required to adapt the behaviour of services, evaluate which will perform best under the given circumstances and to decide when evolution is required. Context information are no part of the Service Request, they are rather a permanent extension that changes dynamically over time.

4.4.1 Context-aware Systems and Related Work

In general, context-aware systems utilize various data to provide services in a context-aware manner. They often consist of components, each responsible for a certain task, e.g. the modelling of context. Baldauf et al. [68] provide a summary about existing context-aware systems. For example, the Context Broker Architecture (CoBrA) [60] incorporates ontologies and inference with the goal of providing context-awareness in smart spaces, e.g. meeting

rooms. In CoBrA a central component, namely the context broker, is responsible for all the tasks related to the context including acquisition, representation, interpretation, and policy management. The context broker communicates with devices, services, and agents in the respective intelligent space in order to update the knowledge base and invoke typical services, e.g. dimming the lights or controlling devices. Another example concentrating on context-aware mobile services is the Service-oriented Context-Aware Middleware (SOCAM) [63], which focuses on the acquisition, discovery, and dissemination of context utilizing a small ontology with general and domain specific concepts. Here, specific context information needs to be actively retrieved by interested services.

In summary, current approaches typically assume a component wrapping multiple context sources that can be queried by external applications in order to retrieve context information. However, this approach turns out to be complicated whenever the involved network environment is unknown and unreliable. Therefore, we propose a model enabling applications to transparently request context information without having to communicate with a centralized context storage component. Furthermore, our approach comprises a bio-inspired collective and distributed knowledge administration and dissemination means complementing the inference of additional knowledge.

4.4.2 Principles of the Context-gathering Overlay

Our ongoing work addresses human-centric service-oriented systems in an open, highly dynamic computing environment. The environment is supposed to have no inherent connection to reliable networking infrastructures. Nevertheless, we regard the seamless supply with context data as a major driver for adaptive services and applications. Based on this view we describe the functional elements of our context gathering architecture and their surrounding environment in the following.

4.4.2.1 Types of Entities

In this architecture the term entity summarizes everything that provides, adds value to, or utilizes context information. The entities are intended to cooperate with each other in order to provide applications with context information on behalf of the user. We first distinguish between three types of entities: Context Sources, Context-aware Services, and Context Relays.

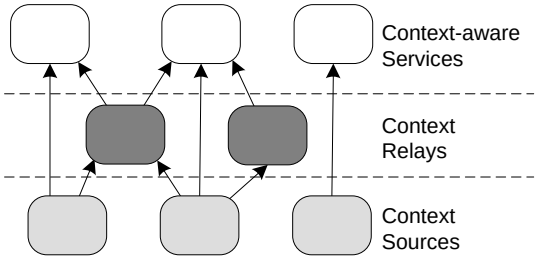

Figure 4-7: Entities and logical flow of context data

Entities providing data which describes the current context of the user are embraced by the term Context Source. Thus, Context Sources are supposed to abstract sensing devices, observing their current physical environment (e.g., temperature, brightness, UV radiation, noise, or pollen saturation), as well as classical information services, (e.g., providing weather information, traffic, or geographical position). Context Sources are considered to be passive. Thus, they provide their information only on demand. In this regard, Context Sources can be regarded as special type of BIONETS Services.

A Context-aware Service is a service that adapts its behaviour according to the current context of the environment or the user(s). A simple example of a Context-aware Service can be found in Scenario 3 of D3.1.1 [81]. Therein, a service searches points of interests for the user, based on the user's current location – a typical kind of context information. Hence, on execution the Context-aware Service tries to gather the required information about user or

environment context. Even certain management components of the BIONETS Service Framework may act as Context-aware Services to obtain environment information, which may help to foresee the need for adaptation before failures occur.

The Context Relay is a supportive, but fundamental element of our approach. Since we consider the computing environment to be highly dynamic and entities to have an alternating visibility, the Context Relay passively caches and reoffers context data once requested. Moreover, Context Relays may ask Context Sources for context information in a pro-active manner to update their caches, as explained later on. Thus, from an architectural point of view, Context Relays behave like a combination of Context Sources and Context-aware Applications. However, in addition to simply caching data, Context Relays may also act as broker insofar as they transform and enrich Context Data on a semantic level - a feature that is especially promoted by ontology-based descriptions. For the moment, Context Relays are intended to be realized as Mediators in the BIONETS Framework, as they cover features that benefit all services on one node and have a special interaction semantic. However, later iterations of this component type may also be realized as Service Cell.

4.4.2.2 Context Spaces

We extended the functional considerations about the entities with further assumptions concerning their distribution. First, Context Sources are supposed to provide their data through short-range radio to devices of people passing by. Therefore, context information is transmitted on demand of the devices, i.e. any Context-aware Service or Context Relay running on them. The accumulation of multiple Context Sources in certain places – for instance places of special interest in a metropolitan infrastructure – forms spots with a high coverage of available data. In contrast, the gaps between those spots are supposed to be undersupplied. Moreover, not each type of information may be found inside all of those spots. Both matters are critical when regarding context-dependent services executed by mobile devices on behalf of the user. For these services the seamless supply with sufficient context information is a basic need. A reasonable approach to compensate the local undersupply with context information is to disseminate context data also apart from stationary Context Sources. Our approach aims to engage entities running on mobile devices for storing and forwarding context information among each other and stationary entities inside different spots.

For that purpose we first introduce the term Context Space for a mobile or immobile ensemble of entities sharing data about the context of a user. The entities composing a Context Space are supposed to be physically close to each other. Since we mainly address Personal Area Networks, the size of a Context Space is determined by physical limitations, i.e. range of radio signals. In the simplest case a Context Space is formed by a single entity, but in general we expect Context Spaces to be composed of multiple entities, spread across different devices. However, as suggested above, mobile Context Spaces are supposed to encounter each other as well as stationary spaces. Hence, the Context Spaces are supposed to be temporally merged when sharing the same physical location. During these periods the composing entities may interact to exchange context data. If these entities are Context Relays, context data are even propagated cross several Context Spaces as depicted in Figure 4-8.

Since context data are highly dynamic they cannot be spread in an arbitrary manner. As argued in [63] they differ in their quality. For instance, the usefulness of location information depends on the distance between the entity that utilizes this information and the source of the information. The same holds for information like temperature or noise. Hence, the quality of context data limits their dissemination in a natural way.

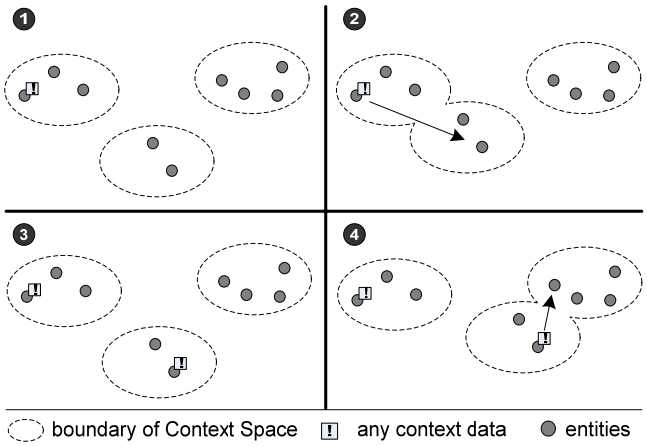

Figure 4-8: Visualization of several Context Spaces and context data forwarded between them

4.4.2.3 Request Driven Context Delivery

In this approach we consider context gathering for highly dynamic, distributed computing environments with a low certainty about reliability and availability of computing components and communication links. Therefore, the interactions among Context Sources, Context Relays, and Services are supposed to be aware of the environment's shortcomings by means of loose coupling. To achieve loosely coupled interactions according to the above characteristics, again the Semantic Data Space is utilized as interaction model. Context gathering is therefore organized in a request-driven manner as illustrated in Figure 4-9.

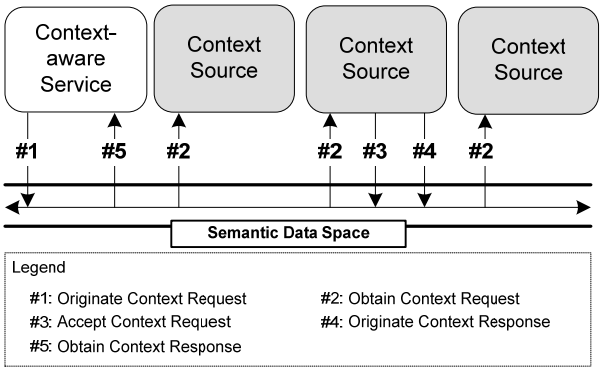

Figure 4-9: Visualization of request-driven context gathering

To obtain information concerning the current context a Service is required to originate an appropriate Context Request and to write this request to the SDS. The request is not required to address its destination. The Context Sources are rather intended to check whether they may provide the requested information or not. For instance, a temperature sensor should accept a temperature request. However, each request should only be accepted by one Context Source. This may be achieved by a first-come-first-serve approach, since the SDS allows each connected party to monitor all exchanged data. The Context Source that accepted the Context Request is supposed to answer immediately by writing a Context Response to the interaction medium that refers to the corresponding Context Request. Disappearing Context Sources can be detected and compensated using timestamps for indicating the expiry of Context Requests and acceptance notifications.

In an open and dynamic computing environment the different terminologies and concepts utilized for context descriptions may be a serious handicap for the communication between Services and Context Sources. Moreover, the alternating availability of Context Sources and consequently certain context information is supposed to negatively influence the Services' behaviour.

In this regard, Context Relays are components that are responsible for constantly gathering, caching, transforming, and reoffering context information, which is provided by Context Sources. Since Context Relays are mobile, the available Context Sources and the structure of the Semantic Data Space constantly change. Therefore, the Context Relay owns a knowledge base to store information gathered in a previous situation. For example, a Context Relay residing on a mobile phone that was able to access a sensor reporting the actual temperature some hours ago is able to make this knowledge available even if the sensor is not accessible any more. Indeed, this kind of knowledge exchange imposes some problems related to the Quality of Context (QoC) [63]. For example, it has to be assured that the provided information is up-to-date or at least accurate to a certain extend. This problem can be solved by linking additional QoC information to the context information, which can be interpreted by the Context Relay, the requesting Service, or even the current user. Another problem is the identification of context information that is important for external Services since the unfiltered storage of data imposes problems related to the available storage and processing time. Regarding this, we developed a mechanism that enables guiding the gathering and distribution of context information based on the weighting of context information. Therefore, context data passed on inside Context Responses is weighted 'inline'. A weight defines how often a certain type of information was requested .That way, we achieve in parallel to the distribution of context data a distribution of relevant values for certain types of information. When receiving weights by listening in other parties interactions Context Relays are supposed to adapt their behaviour of renewing or deprecating certain context information accordingly.

Figure 4-10 shows an example with the Context Relay involved in the communication process providing additional knowledge. Whenever context information is requested using the Semantic Data Space the Context Relay is notified. Since the requested information cannot be provided by the two Context Sources subscribed to the Semantic Data Space, the Context Relay tries to provide the lacking information searching its local knowledge base after a certain period of time.

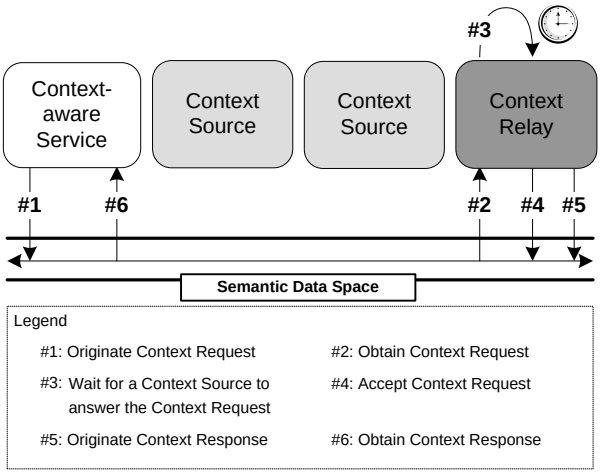

Figure 4-10: Request handling supported by the Context Relay

5. Service Self-Management

5.1 Introduction

The aim of this section is to analyse, what kind of autonomous functions are required by the BIONETS services and also to give a short description how these functions could be achieved in the BIONETS computing environment. First we will research issues related to service self-control, e.g. how services manage and control themselves in a BIONETS service environment and we also research service autonomic lifecycle management related issues.
The autonomic functions of the BIONETS services (In this section the term service can be either Service Cell or Service individual) can be implemented either inside of the service or offered by the hosting node in a form of Service Mediator. In this section we don't differentiate, which case is it, because i.e. in case of Service Individual the Service Mediator logically can be seen also as a part of the Service Individual.

The essence of autonomic computing systems is self-management, which intent is to free the users from the details of system operation and maintenance. In the BIONETS environment service self-management implies some support for autonomous decision-making inside the services. These autonomic services will maintain and adjust their operation in the face of changing components, workloads, demands, and external conditions.

The self-management procedures in an autonomic system can exist in many levels. At first, automated functions can merely collect and aggregate the available information to support decisions by users. Or, procedures can serve as advisors, suggesting possible courses of action for users to consider [23]. In some current systems the autonomic systems can even make some lower-level decisions on behalf of the user. The purpose of the BIONETS system is to take the autonomic procedures even further, where the users will need only to make relatively less frequent predominantly higher-level decisions, which the system will carry out automatically via more numerous, lower level decisions and actions.

Traditional service-oriented architectures like Web and grid services [24][25] can be used as foundations of future autonomic service environments, but they don't answer all the problems in a sufficient way. For example, when acting as a service provider, the Service Cells will not honour requests for service unquestioningly, as would typical Web service do. The autonomic service will provide a service only if it is consistent with their "higher" goal or individual benefit. Also, as consumers, autonomic elements will autonomously and proactively issue requests to other elements to carry out their objectives. Another difference with the traditional service-oriented concepts is that the autonomic elements will have complex life cycles, continually carrying on multiple threads of activity, and continually sensing and responding to the environment in which they are situated [26].

In a BIONETS service execution environment there is a paradigm that everything should be a service. Some services can be offered to customers, such as Voice over IP (VoIP), while others are used as components to build other services, such as IP packet transport. Some services in the BIONETS environment are "atomic", i.e. cannot be broken down into component services anymore, and usually act on the underlying resources. As introduced in earlier sections, these atomic services are called Service Cells and compositions of these Service Cells are called Service Individuals.

In the BIONETS environment, the number of devices and also the number of services running on those devices exceeds the number of human users by several orders of magnitude, making manual configuration and management unrealistic to implement. The future BIONETS service environment is dynamic, heterogeneous and resource constrained, therefore in order to be able to survive and evolve in such environment without relying on a centralized control the BIONETS service self-control function needs to be able to automatically manage, optimize, monitor, repair, protect and evolve itself. We have to also carefully consider how self-aware

the service must be in order to create better performance to the service, but still keep the service complexity reasonable.

When analysing in a more detailed manner what kind of requirements service self-control/self-management functions offer to BIONETS services, the following functionalities emerge:

- *Service Creation:* One of the key aspects of service self-control is the automatic creation of services. A Service gets the "goal" from the user or detects that some new functionalities are needed and automatically translates these to lower level requirements and finds the services that provide the needed functionality. A service creation can be triggered on demand, e.g., by a request for a certain task that cannot be solved be a currently existing service, or in self-evolving manner, to provide the user system-driven with new capabilities. Services use the fitness values to choose a best available service offering a specific functionality among many similar services. Service creation is supposed to be based on the composition of basic service building blocks. These service building blocks may be services themselves.

- *Self-aware and context aware Service Provisioning:* Nodes should be aware of the current state of their own services, but also the current state of remote services and they should be able to adapt their behaviour accordingly, e.g. do not try to ask for a service which is already busy.

- *Self-awareness:* An autonomic service should also be aware of its current state and should have enough information to fulfil its purpose.

- *Self-Managing & Self-adaptation:* The service should automatically detect the meaningful changes. For example, if the service migrates from one node to another, the service should adapt itself to changed conditions. The service should also be capable of managing its own resources.

- *Environment-awareness:* An autonomic service must know its environment and the context surrounding its activity, and act accordingly.

- *Self-optimization:* BIONETS service should always look for ways to optimize its workings and seek opportunities to improve its own performance and efficiency.

- *Self-configuration:* In BIONETS system the service should configure and reconfigure itself under varying unpredictable conditions in accordance with high-level policies.

- *Service Execution*: The service should be able to control its own execution in a self-reliant manner. I.e. the service should be able to relieve, recover, and restart itself. The service should also be able to handle execution requests from other nodes, i.e. service could execute also services in other nodes.

- *Self-protection:* BIONETS service must detect and identify various types of attacks and protect itself against them to maintain overall system security.

- *Self-monitoring:* BIONETS service should be capable to monitor itself and try to detect possible software and hardware problems as they appear.

- *Self-repair:* BIONETS service should be able to repair itself after detecting problems

Incorporating autonomy on Service-Level would make the services really autonomous, which would be quite good from the scientific point of view, but in case of multiple Service Cells on the same node in the same time it would result in the duplication of functionality and resource consumption (overhead due to the management stuff integrated in each Service Cell). Another approach would be incorporating autonomy on the Node-Level. This would result with no

duplication of cooperation-related processes on one node. Also the services are kept simple and may also be provided by T-Nodes. Security may also benefit from the decoupling of framework-related tasks and the functions provided by Service Cells. Another benefit is that because the services are simpler therefore less effort is required when implementing services. Downside of this approach would be the classic appearance of services which would not be autonomous themselves.

5.2 Autonomic Lifecycle Management

One of the aims of the BIONETS project is to research the issues related to the automation of the BIONETS Service Life Cycle i.e., the life span of services including creation, deployment, evolution, and retirement aspects.

An autonomic service's life cycle begins with its design and implementation; continues with test and verification; proceeds to deployment, installation, configuration, optimization, upgrading, monitoring, problem determination, and recovery; and culminates in un-installation or replacement. Each of these stages has special issues and challenges: [23][27]

- *Design and implementation:* The design and implementation phase can be divided to two cases: "normal" service creation by user or autonomic service creation. The BIONETS service doesn't necessarily need to be created/deployed by the user, but it can be also automatically created and deployed. For example, the BIONETS system could recognize the needs of the user by using the user's contextual or history information and see that the user will need a specific service.
- *Service composition:* A Service gets the "goal" from the user or detects that some new functionalities are needed and automatically translates these to lower level requirements and finds the services that provide the needed functionality. A service creation can be triggered on demand, e.g., by a request for a certain task that cannot be solved by a currently existing service, or in a self-evolving manner, to provide the user system-driven with new capabilities. Services use the fitness values to choose a best available service offering a specific functionality among many similar services. Service creation is supposed to be based on the composition of basic service building blocks.
- *Testing:* Testing of autonomic services will be particularly challenging because of the complexity of the environment…
- *Deployment, installation and configuration:* After the service is created and tested it will need to be deployed/installed and configured in the new location. E.g. [29][30]
- *Optimization, upgrading, monitoring, problem determination, and recovery:* After the service is installed and configured it will have to control itself in order to survive and function the best possible way in the new…
- *Service migration:* A Service moves/copies itself automatically to another U-node. E.g. [31]
- *Evolution:* Service evolution is utilized at three points in the service life cycle: 1) service creation, 2) service binding, 3) code generation.
- *Deprecation, un-installation or replacement:* Services are expected to have limited span of life, they are for instance deprecated if they are not used any more.

Autonomous properties and behaviors of services build the basis for service self-management and control, implying a bipartite integration approach. On the one hand, there are components addressing single aspects within the BIONETS service life cycle; they are therefore directly interlaced within the life cycle and have already been introduced in section 3. On the other hand there are some components appearing conceptually orthogonal to the BIONETS service lifecycle, influencing the life cycle holistically. These components mainly base on

collaborative management issues and service self-awareness; they are discussed afterwards in section 5.3. The two-part realization of service self-management and service self-control is depicted in Figure 5.1 with regard to the prior established concept of BIONETS service life cycle.

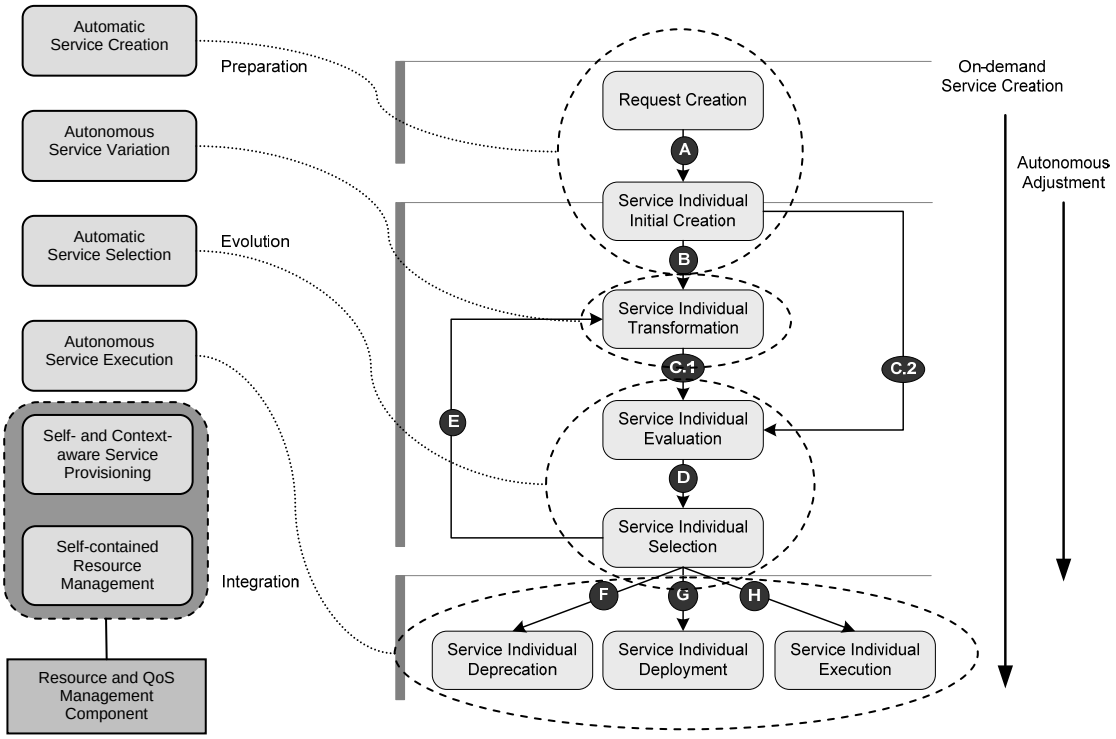

Figure 5-1: Service Self-Management Integration

5.3 Resource Management

5.3.1 Service Request Driven Cooperative Resource Management

Since services and nodes are meant to be self-aware and self-contained, explicit resource management is regarded to be an issue for every single node. Thus, a node is not allowed to allocate remotely a resource of another node; the resource can only be directly accessed by the node possessing the resource. Thereby, collaborative resource management is demoted to local resource management.

This local resource management can be divided into two main parts. One the one hand, there are requests that can be treated self-reliantly by the node that has received the request with services and resources locally available. In this case, common resource allocation and scheduling principles can be used in order to provide an efficient and fair resource management.

On the other hand, there are requests that can only be handled in a collaborative manner. As introduced in section 4.2.4, requests are posed on a shared medium (like a SDS) where every participating service can subscribe to a subset of this request it can provide a solution for. The subscription of a service to a sub-request however influences the resource management and Quality of Service (QoS) capabilities of the node executing this service. In other words, the subscription to a sub-request directly corresponds to a resource allocation provided by the host node of the service.

Thus, although a node cannot allocate a remote resource, it can request a remote service execution accessing this resource. The hardware related resource management of single nodes

is thereby extended to a service based resource sharing where the resource management itself is transparently encapsulated into a service provisioning procedure.

For instance, a node requires its GPS coordinates. Instead of providing a resource management mechanism that allows the node to allocate a GPS device from a node within the network, we envision to expand the request driven approach introduced in section 4. Thus, the node poses a request indicating its need of its current GPS coordinates. Another node within the network possessing a GPS device may agree to provide the requesting node with a solution. The workflow is abstractly depicted in Figure 5-2. A service requiring resources the node it is executed on does not obtain itself in order to cope with the user's request poses a Service Request to a Shared Space accessible for all nodes within the current network (1). Services can read Service Requests from the Shared Space and decide whether they are willing and able to provide help for some sub-request (2). If a service decides so, it calculates the according result with its locally available resources (3). It is afterwards transferred to the Shared Space (4) where it can be accessed by the service that has initiated the Service Request (5). This communication procedure, containing a request announcement, a request subscription and a solution provisioning, can be implemented via the same data space as already introduced in section 4.2.

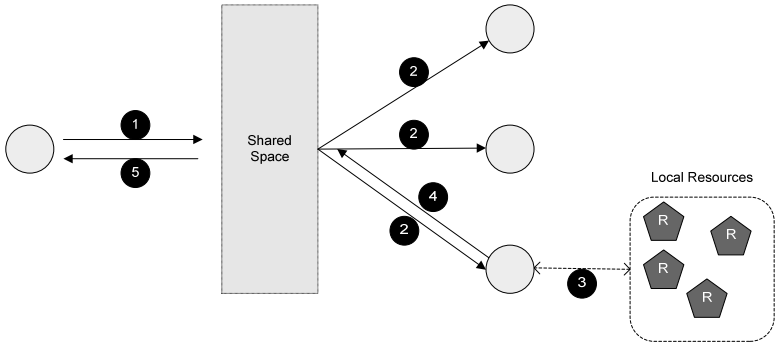

Figure 5-2: Workflow for Service based Resource Management

Thus, resource management in BIONETS covers the provisioning of scarce resources (like a GPS device) through service requests. Therefore, resource management procedures enable users (i.e. nodes) to cope with requests they would not be able to deal with self-reliantly because of missing information. This information can be provided by other nodes within the network possessing resources that are capable of providing/gathering this information. In these terms, collaborative resource management overcomes limits in request processing abilities of nodes in functional terms rather than in non-functional ones, i.e. provides solutions for enabling nodes to cope with requests they would not be able to respond self-reliantly. Based on this view, T-nodes (such as temperature sensors etc.) can be somehow regarded as such spare resources being able to subscribe to tasks commonly unsolvable for mobile user devices.

Notably, this approach is also capable of dealing with problems addressed by Ad Hoc Grid Computing [75][76], i.e. the conglomeration of resources like CPU cycles and memory allowing even devices with less computation power to execute computation intensive applications by dint of nodes with higher computing capabilities. However, the focus of BIONETS resource management will remain on scarce resources, assuming nodes to be powerful enough to deal with user request self-reliantly in terms of pure processing power.

The introduced approach implies the insertion of an additional resource management component enabling the user (i.e. the single node) to remain in control of its own device.

5.3.1.1 Resource Management Evaluation

Services may be assumed to subscribe to every sub-request they are able to cope with. This autonomous behaviour would however heavily influence the resource availability and assurance of QoS capabilities of nodes hosting powerful or frequently required services, since a great amount of their local resources is required for computing solutions for subscribed sub-

requests from other nodes. Thus, a component is required restricting services in their subscription behaviour in order to provide a certain level of fitness in terms of resource availability for every node; thereby, the user remains in control of his or her device.

We therefore introduce a Resource Management Evaluator (RME) deciding whether a service is allowed to subscribe to a certain sub-request it is able to provide a solution for or not. As depicted in Figure 5-3 the RME consists of four evaluation components, a resource repository containing a list of resources available on the single node as well as their specific capabilities, a communication interface and a rule repository.

1. *Service Request Requirements Evaluation:* The hardware requirements for solving a specific request have to be regarded. It is envisioned that those are estimated by the evaluation component. In order to enhance the accuracy and speed of this procedure, semantically descriptions may be additionally attached to sub-requests providing a list of required resources. As motivated above, the evaluation of resource requirements will mainly focus on the eventual allocation of scarce resource rather than on calculation-enabling resources like CPUs and memory. However, it has to be regarded that even those resources can become bottlenecks assuming a high number of remote service subscriptions on one node.
2. *Requester Evaluation:* The subscription to a sub-request may depend on the user or device that has posed the request to the SDS. There may be certain groups agreeing to aid themselves in a special manner, friend-of-a-friend principles or other community-oriented approaches influencing the decision of overtaking a sub-request. Moreover, a user may pay for a service (i.e. a fix installed node only provides the user with a service execution if he or she has registered and is willing to pay for it).
3. *Context Evaluation:* Since the context can heavily influence the performance and the resource requirements of context-aware services, the current surrounding of the node can additionally be taken into consideration when evaluating a possible sub-request subscription.
4. *Scheduling Evaluation:* A node should be able to predicate its subscription on both its current state and its assumed future situation. For instance, a node may have learned that it will eventually face high resource allocations by the user and therefore opt not to subscribe to a certain sub-request in order keep the resources available for the future local user requests. As for the Service Request Requirements Evaluation, the eventual scheduling and allocation of scarcer resources will dominate this evaluation procedure.

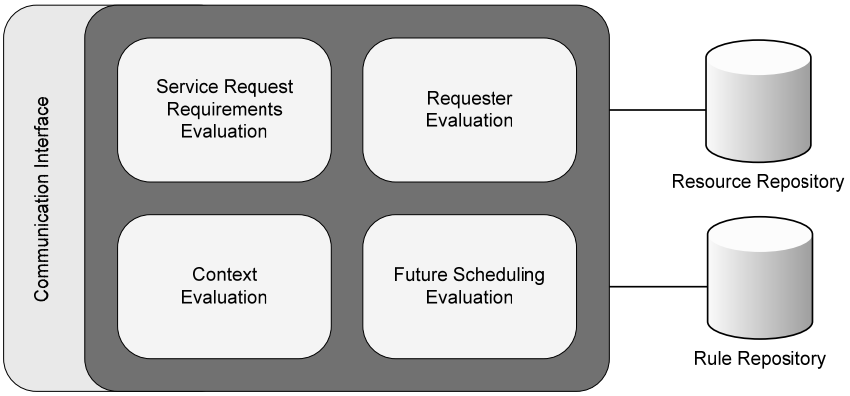

Figure 5-3: Resource Management Evaluator (RME)

5.3.1.2 RME applicability and ease of integration

These prior introduced evaluation components are regarded to build the basis for resource based fitness evaluation of nodes. It is however envisioned to design an interface enabling other evaluation components to be plugged into the RME in order to enhance its evaluation capabilities, scope of applicability or accuracy.

RMEs can be restricted to operate on selected resource only. For instance, a user holds a set of resources r_1 to r_7.as depicted in Figure 5-4.He or she may decide to share resource r_1, r_2 and r_3 without restriction within his or her community, but not with other users. Resources r_4 and r_5 are completely forbidden to be accessed by services trying to subscribe to a remote sub-request while resource r_6 and r_7 are released for restricted allocation in a way that at least 60% of their capabilities have to remain reserved for the user. This knowledge is held as a set of rules within the rule repository. Rules can be assigned explicitly by the user, deduced by his or her profile and learned with regards to the user's current and foreseen behaviour and context.

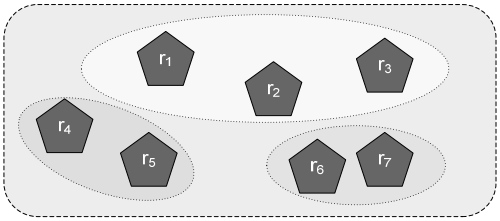

Figure 5-4: Possible categories implied by resource provisioning rules

Moreover, RMEs are not restricted to evaluate the resource situation of single nodes only. They can be furthermore assigned to manage the resource of a set of nodes, e.g. an ad hoc community. This might be especially interesting for a group of users sharing all their resources without any restriction. Thus, it may be sufficient to ensure that predefined level of resources and resource capabilities is reserved for the community, independently from the physical location of this resource. This configuration would solve a set of problems arising due to hidden terminal issues. Imagine 4 devices d_1 to d_4 sharing their resources in a way as outlined above, freeing 30% of their calculation power for other users. Either because of its physical position or its service availability, device d_1 receives much more subscription requests than the other devices. With one central community RME, it is possible to free nearly most of the calculation power of d_1 for remote request subscriptions, because the remaining devices within its community are able to compensate the higher load of d_1. If all devices were equipped with an own RME, d_1's resource provisioning would have been restricted to 30% as predefined.

As introduced, the RME aims at providing some control functionalities to the user of a device. There are mainly two realization possibilities for this component. It may be either implemented on every node requiring the introduced control functionalities or within every service. Since the control represents a functionality provided to a user, one RME per node seems to be sufficient. However, this component thereby would represent a very central component services are dependent on. An integration within every service would enable a higher level of autonomy at the expense of the number of redundant copies of the control component. Moreover, the integration within every service will inflate the services' size significantly and probably prevent them from being executed by devices with less computing capabilities.

We therefore envision to initially implement the RME centrally on every node in order to enable a more light-weight design of services. In a refinement step, we will try to move the control component from node to service level in order to aim for more autonomous service behaviour. This proceeding is chosen according to the pursuit increase of service autonomy as illustrated in Figure 2-1.

The communication interface (see Figure 5-3) therefore services two communication requirements. Assuming RMEs to be implemented centrally on nodes, these RMEs will have to exchange information in order to virtually create a "community RME" as introduced above. When distributing the component on every service, the single RMEs have additionally to communicate among each other in order to align their striven sub-request subscriptions and to collaboratively build up a coherent picture of the current status of the node or the community.

Although we have introduced the RME to restrict the subscriptions of services to sub-requests, it remains to be clarified why a service should subscribe to sub-requests (and thereby consume local resources) at all. First, one could argue that every user is willing to provision computing capabilities to other nodes so it can itself use the capacities of other nodes; this principle was successful in multiple peer-to-peer based file-sharing clients [72][73][74]. It is additionally envisioned to incorporate different reward functionalities for participating within the BIONETS community. It could be possible to reward the level of contribution to the BIONETS community, i.e. the computing capabilities that are provisioned for a node are proportional to the ones offered by it (with regard to the overall computing capacities of the node). More reward principles are envisioned to be derived from D3.2.3 dealing with possible business models for BIONETS.

5.3.1.3 Solution Proposal Evaluation

As already motivated, the striven resource management approach enables users with restricted resource availability to perform operations they would not be able to cope with self-reliantly by aid of other nodes. In order to enhance this procedure, not only a method restricting services from subscribing although the current node they are executed on is already strongly loaded, but also a method for selecting appropriate nodes for executing sub-requests is required. First, this decision making can influence the request processing in performance aspects, for instance some nodes may be able to provide solutions faster than other ones. Second, the quality of the possible information could be evaluated, depending on the state of the requesting node and the context of the node supposed to contribute to the sub-request. For example, a node requires its GPS coordinates. A portable GPS device as well as a fixed one (i.e. possibly a T-node) may overtake this sub-request. The node may now decide, which coordinates may be the most accurate ones, depending on the location of the remote nodes, the actuality of the coordinates etc. Thus, a node that has provided a request to the SDS should decide, which node executing a service able to compute a result for a sub-request should overtake this task.

We therefore envision a proposition and subscription based mechanism operating on a SDS; the related interaction is abstractly depicted in a sequence diagram in Figure 5-5. Since the SDS is not an object itself, it is not represented within the sequence diagram; however, the multiple messages sent from the requester to the nodes and vice versa within the current network have to be understood as a communication via a SDS as introduced in section 4. Thus, the requester pushes its requests to the SDS where it is read by other nodes within the network. Services able to subscribe to a request are evaluated by a RME (either implemented within the service or the node) whether their subscription is compatible with the resource management and QoS demands of the user. If yes, the RME has to appraise under which circumstances the sub-request can be treated. These circumstances may be the overall time required for computing a solution, its foreseen accuracy and fault probability etc. Notably, this evaluation procedure is performed by the RME with regard to the requester, the request requirements, the current context and the foreseen scheduling of the node. The estimation is encapsulated into a *Solution Proposal* that is pushed back to the SDS. By sending a Solution Proposal, the node agrees to reserve the resources necessary to eventually calculate the according result in case it is selected to do so. After the requester has either waited for a self-chosen time span or obtained a sufficient amount of Solution Proposals from the SDS, it evaluates the proposals and acknowledges the one which seems to be the most suitable one. Presumed the situation of the node that has received this ACK has not significantly changed due to other service subscriptions in the meantime, the service now subscribes to the sub-request whose according proposal was accepted by the requester; the requester confirms this subscription with another ACK. It moreover sends NACKs to all nodes whose Solution Proposals have not been accepted. These nodes can thus release the resources they have been reserved for the case that they are assigned the sub-request and are free to reserve them for future Solution Proposals. Notably, these NACKs are necessary in order to prevent nodes to become easy targets of Denial of Service attacks. Moreover, nodes posting Solution Proposals

should set timers so they can annul their resource reservations after a maximal waiting time has been elapsed; these timers can prevent nodes from malicious requesters never sending any ACKs or NACKs for received Solution Proposals. When receiving the 2nd ACK confirming the reception of the subscription message from the requester, the chosen node begins to compute the solution for the sub-requests and pushes it to the SDS when finished where it is finally received by the requester.

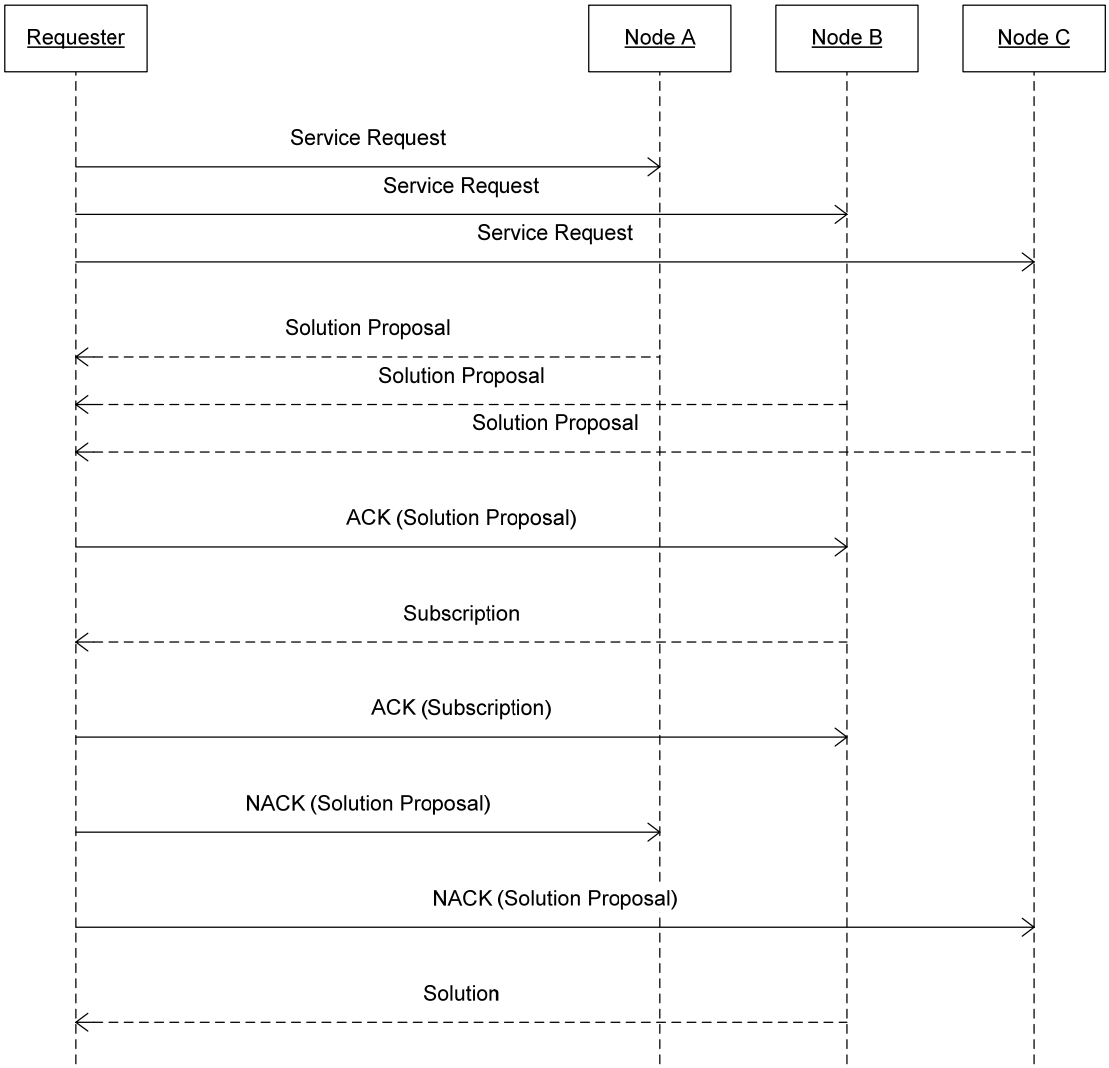

Figure 5-5: Proposal based Service Request Subscription

The integration of RMEs within services requires a certain level of self-awareness of services themselves. Thus, a service has to be aware of its current status in order to decide whether it is capable of subscribing to a certain sub-request or not. Moreover, services have to communicate their states to form a coherent picture of the current and future node workload and status. Section 5.3.2 therefore introduces *service states* providing a solid basis for a self-contained service design.

5.3.2 Service life-cycle state diagrams

In the service life-cycle state diagrams presented below we can see different internal states of BIONETS services (both Primary Service Cells and Service Individuals). The services should be intelligent enough to be aware of its purpose, should have enough information to fulfil its purpose, have certain degree of self-awareness and be aware their internal state and also internal states of other subcomponents in the same Service Individual or node. It is also

possible that the service can see even states of services locating on nearby nodes. Using this information the services should be able to adapt their own behaviour, e.g. do not try to ask for a service which is already busy. The service can control its own life-cycle through changing its internal state; the trigger for a state change could for example be a change in the node resource environment or network connectivity. The trigger to start state transition could also originate from outside the service. For example, in case of node level autonomy the node's Service mediator could control the state of the service, thus enabling the autonomic behaviour of the service. It could also be envisaged that the other services could control others service states, e.g. the higher-level service could have control over the other service in the same service composition.

In order to survive in the BIONETS environment it is not enough for a service to know its environment and the context surrounding, but the services should also act accordingly. If the service migrates from one node to another, the service should adapt itself to changed conditions. It should be able to recover, restart and relieve autonomously and e.g. in case of changes in the environment the service could decide self-reliantly to evolve.

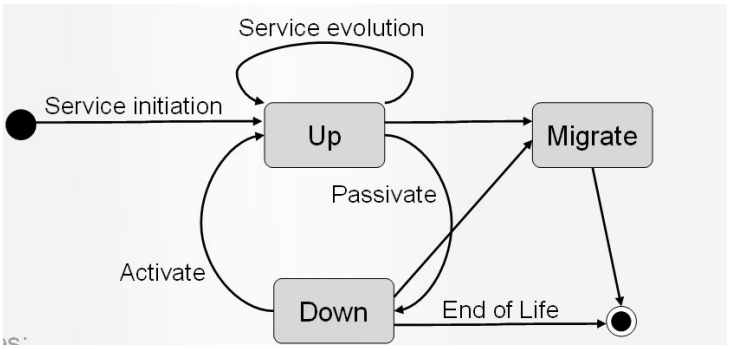

Figure 5-6: The top level states of the Service life-cycle

States:

- Up — the service is capable of accepting and processing requests (i.e. the service is available).
- Down — the service is not capable of accepting any requests (i.e. the service is not available).
- Migrate — the service is migrated/duplicated to other node (i.e. the host node is shutting down, lack of resources, user request,…).

Transitions:

- *Service initiation* — the service starts its life in Up state.
- *End of Life* — the service ends its life from Down state.
- *Activate* — the service can become available which transitions it from Down to Up state.
- *Passivate* — the service can become unavailable which transitions it from Up to Down state.
- *Service evolution* — Service evolution based on changed user requirements and environmental changes.

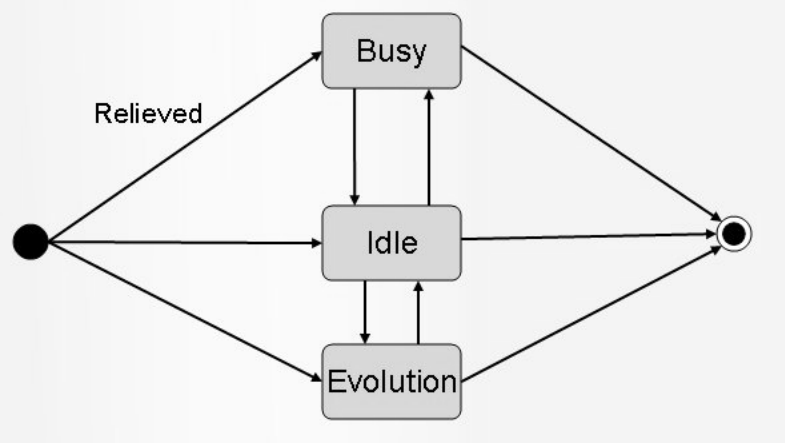

Figure 5-7: Up-substates of the Service life-cycle

States:

- *Idle* — the service is not processing any requests currently.
- *Busy* — the service is processing requests currently.
- *Evolution* — the service is changing through the evolution.

Transitions:

- A service normally enters Up state in Idle state.
- *Relieved* — a service enters Up state in Busy state if it was relieved (from Down/Saturated).
- A service may transfer from Idle to Busy and vice versa.
- A service exits Up state from either Idle or Busy.

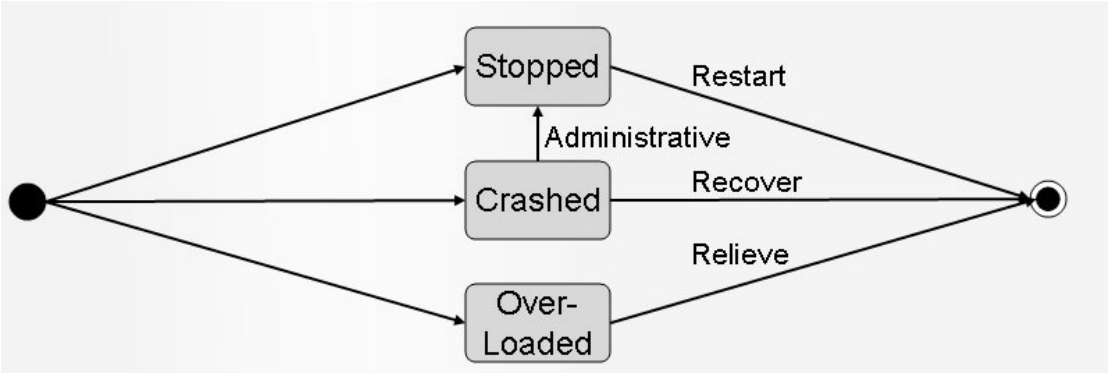

Figure 5-8: Down-substates of the Service life-cycle

States:

- *Stopped* — the service was intentionally stopped (e.g. for administrative purposes).
- *Overloaded* — the service has exhausted its resources and cannot accept any new requests.
- *Crashed* — the service is unavailable because of an internal malfunction of the service or environmental problem (e.g. external problem in hosting node).

Transitions:

- A service enters Down state in either a Stopped, Crashed or Saturated state.
- A service may be stopped from any Up or Down state.
- *Relieve* — a service is relieved from overload and exits Down state (to an Up/Busy state).

- *Recover* — a service was recovered from malfunction and exits the state. (to an Up/Idle state).
- *Restart* — a service was started again (manually) and exists the state. (to an Up/Idle state).
- *Administrate* — a service can be stopped from being crashed for administration and maintenance.

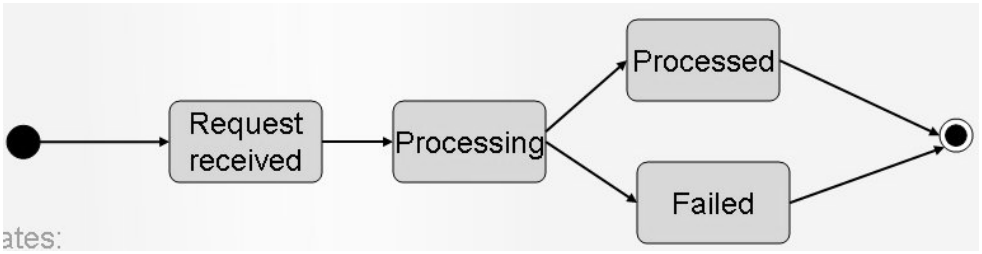

Figure 5-9: States of the Service request processing

States

- *Request received* — the service request and/or change (from the user or environment) indication is accepted.
- *Processing* — request processing/execution for example self-management, self-healing, mutation are performed if needed.
- *Processed* — the service has successfully completed requested function returning results to the requester.
- *Failed* — the service encountered an error and didn't complete the requested function, returning error to the requester.

Transitions

- A service starts request processing when it accepts (receives) a request.
- A service starts execution (actual processing) after it received a request.
- A service transitions to either processed or failed state depending on the outcome of the processing/execution stage.
- A service exits request processing from either processed or failed state (which are mutually exclusive according to the previous transition).

6. Conclusion and Outlook

This deliverable is regarded to provide a basis and orientation for further research on of bio-inspired services, thus delineating a starting point for building up the BIONETS service architecture. The BIONETS service life cycle is considered to be the centerpiece of future research, while its different aspects are currently still differing heavily in their level of progression. The further development of the current state of the BIONETS life cycle is already in progress as deliverable D3.3.2 deals with business models which are assumed to contribute to the service self-management and deliverable D3.2.2 focuses on service evolution, eventually providing additional contributions to service evolution, evaluation, selection and deprecation aspects within the BIONETS service life cycle.

A main objective of WP3.2 centers on service autonomy. As illustrated in Figure 2-1, we propose a shift of autonomy integration from the execution environment to the services themselves. Thereby we obtain a working version in the first instance while coevally allowing further investigations in order to come closer to the long-term goal of completely autonomously acting service environments.

In detail, the different components of this deliverable are considered to be in a progression state as follows: The Service Framework builds upon a mediator controlled architecture. In the first instance, the mediator components are inserted per node, enabling a node-central service management, thus a light-weight service design. The future work will center on integrating the mediator component into the services so they are able to act completely self-contained and autonomously. It has especially to be investigated in which way this shift of complexity influences both the service feasibility and gain in functionality.

As the entry point for the BIONETS service life cycle a formulation of a user request within a natural, user-friendly language is envisioned, which is then transacted into a service composition capable of computing an according reply. This objective is also aimed to be reached in multiple steps, where the earliest centers on a service composition manually performed by the user via a graphical interface. In further refinement steps, it is aspired to incrementally free the user from conceptual contributions to the solution of the posed request by introducing autonomous service composition procedures collaboratively performed by multiple nodes within the current network. For the automatic creation of Service Individuals (i.e. service compositions), multiple paths will be explored in the future, including component based, state-based and semantically enhanced approaches. The automatic transformation from a user request to a service request representing the possible service compositions is finally envisioned to enable the user to specify *what* should be achieved instead of *how*. The resulting Service Individuals representing possible service compositions fulfilling the request with various levels of exactness both in terms of functional and non-functional properties are optionally transformed by evolutionary processes. This topic has only been covered slightly so far since is addressed in Task T3.2.4 and therefore will be elaborated within deliverable D3.2.2. For service deployment, multiple approaches are pursued. The work has so far centered on the migration of complete service compositions, i.e. code migration. It is moreover aimed at investigating deployment techniques for Service Individuals, with other words, of plans of possible service compositions. Since most challenges resulting from the implementation of these deployment procedures are assumed to depend significantly on the representation of Service Individuals, they will be addressed in more detail within the next month when the picture of Service Individual representation becomes clearer.

While the interaction framework is within an advanced state already including a running implementation, the self-management concepts of services still reside in an early phase. As already mentioned above, some results from deliverable D3.3.2 dealing with business models are envisioned to contribute to the management of the participation of nodes in remote requests scenarios. Moreover, as pointed out for the service framework, the resource management and quality of service evaluation component will be integrated centrally within

nodes in order to provide an easy implementation for the first instance, and is then supposed to be moved within services enabling them for complete autonomy and self-control.

In summary, the results achieved within this deliverable already provide a reliable and substantial fundament for BIONETS services and service architecture. Most aspects are tackled in bottom-up manner with regard to autonomy, i.e. aim at establishing functional correct solutions in early phases of the development at the expense of an undesirable, but manageable degree of user-interaction, centrality and inflexibility. Those concessions are then pursued to be eliminated by further refinement steps, paving the way to a completely autonomously acting architecture.

7. References

[1] D. Barkai. Internet distributed computing the intersection of web services, P2P, and grif computing. In P2P'02, 2002.
[2] P.. Bellavista, A. Corradi, and S. Vecchi. An Integrated Resource Management Architecture for Wireless Smart Environments. In WOA 2003: 4th AI*IA/TABOO Joint Workshop "From Objects to Agents": Intelligent Systems and Pervasive Computing, pages 49–56, Villasimius, CA, Italy, 2003. Pitagora Editrice Bologna.
[3] R. Buyya and S. Venugopal. A Gentle Introduction to Grid Computing and Technologies. Computer Society of India, July 2005.
[4] D. de Roure, M. Baker, N. Jennings, and N. Shadbolt. Grid Computing - Making the Global Infrastructure a Reality, chapter The evolution of the Grid, pages 65–100. John Wiley and Sons Ltd., 2003.
[5] W. Dou, Y. Jia, Z. Liu, and P. Zou. A Simple Constructing Approach to Build P2P Global Computing Overlay Network. In ICANN 2003, 2003.
[6] L. Ferreira, V. Berstis, J. Armstrong, M. Kendzierski, A. Neukoetter, M. Takagi, R. Bing-Wo, A. Amir, R. Murakawa, O. Herndandez, J. Magowan, and N. Bieberstein. Introduction to Grid Computing with Globus. IBM, imb.com/redbooks, 2003.
[7] I. Foster. What is the Grid? A Three Point Checklist. Argonne National Laboratory & University of Chicago, July 2002.
[8] T. Friese, B. Freisleben, S. Rusitschka, and A. Southhall. Framework for Resource Management in Peer-to-Peer Networks. In Proceedings of the International Conference NetObjectDays, pages 4–21, Erfurt, Germany, 2002. LNCS 2591, Springer.
[9] T. Friese, J. P. M¨uller, M. Smith, and B. Freisleben. A Robust Business Resource Management Framework Based on a Peer-to-Peer Infrastructure. In Proceedings of the 7th International IEEE Conference on E-Commerce Technology, pages 215–222, Munich, Germany, 2005. IEEE Press.
[10] Globus. The Globus Alliance Homepage. http://www.globus.org/alliance/, 2006.
[11] Globus. Grid Resource Allocation & Management Protocol (GRAM). http://www.globus.org/toolkit/docs/3.2/gram/ws/, 2006.
[12] Globus. Monitoring and Discovery Service (MDS). http://www.globus.org/mds/, 2006.
[13] J. Liu, Q. Zhang, W. Zhu, J. Zhang, and B. Li. A Novel Framework for QoS-aware Resource Discovery in Mobile Ad Hoc Networks. In IEEE International Conference on Communications (ICC'02), pages 1011 1016. IEEE Computer Society, New York City, USA, April 2002.
[14] P. Plaszczak and J. R. Wellner. Grid Computing: The Savvy Manager's Guide (The Savvy Manager's Guides). Morgan Kaufmann, August 2005.
[15] S. Rusitschka and A. Southall. The resource management framework: A system for managing metadata in decentralized networks using peer-to-peer technology. In Lecture Notes in Computer Science 2530, pages 144–149. Springer, 2003.
[16] R. Schollmeier. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01), pages 27–29, Los Alamitos, CA, USA, 2001. IEEE Computer Society.
[17] S. Shi, J. Li, C. Wang, Y. Wu, and B. Liu. Energy-Efficient Adaptive Resource Management Strategy for Large-scale Mobile Ad Hoc Networks. In Proceedings of the The Fifth International Conference on Computer and Information Technology, pages 410–416, Washington, DC, USA, 2005. IEEE Computer Society.
[18] I. Stoica, R. Morris, D. K. andF. Kaashoek, and H. Balakrishnan. "Chord". In ACM SIGCOMM, 2001.
[19] Sun Microsystems. Java Management Extensions (JMX). http://java.sun.com/products/javamanagement/doc.html, 2006.
[20] D. Tang, C. Chang, K. Tanaka, and M. Baker. Resource Discovery in Ad hoc Networks. Technical report no.: Csltr-98-769, Stanford University, August 1998.
[21] J. B. Tchakarov and N. H. Vaidya. Efficient Content Location in Wireless Ad Hoc Networks. In IEEE International Conference on Mobile Data Management (MDM'04), pages74–87, Berkeley, California, January 2004. IEEE Computer Society.
[22] P. Uppuluri, N. Jabisetti, U. Joshi, and Y. Lee. P2P Grid: Service Oriented Framework for Distributed Resource Management. In Proceedings of the 2005 IEEE International Conference on Service Computing (SCC'05), 2005.
[23] Kephart, J.O. and D.M. Chess, The vision of autonomic computing. Computer, 2003. 36(1): p. 41-50.
[24] Kreger, H., "Web Services Conceptual Architecture," v.1.0. 2001; http://www-4.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.
[25] Foster I. et al., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Feb. 2002; http://www.globus.org/research/papers/ogsa.pdf.
[26] Farha, R. and A. Leon-Garcia. Blueprint for an Autonomic Service Architecture. In Autonomic and Autonomous Systems, 2006. ICAS '06. 2006 International Conference on. 2006.
[27] Herrmann, K., G. Muhl, and K. Geihs, Self-Management: The Solution to Complexity or Just Another Problem? IEEE Distributed Systems Online 2005 6 (1): p. 1.
[28] Chiang, F. and R. Braun. A Nature Inspired Multi-Agent Framework for Autonomic Service Management in Ubiquitous Computing Environments. in Computational Intelligence Methods and Applications, 2005 ICSC Congress on. 2005.
[29] Saffre, F. and H.R. Blok. "SelfService" - a theoretical protocol for automatic distribution of services in P2P communities. in Autonomic Computing, 2004. Proceedings. International Conference on. 2004.
[30] Haas, R., P. Droz, and B. Stiller, Autonomic service deployment in networks. IBM Systems Journal, 2003. 42(1): p. 150-164.
[31] Suzuki, J. and T. Suda, A middleware platform for a biologically inspired network architecture supporting autonomous and adaptive applications. Selected Areas in Communications, IEEE Journal on, 2005. 23(2): p. 249-260.
[32] Shenker, S.J., Making greed work in networks: a game-theoretic analysis of switch service disciplines http://dx.doi.org/10.1109/90.477727 IEEE/ACM Trans. Netw. , 1995 3 (6): p. 819-831.
[33] Srinivasan, V., et al. Cooperation in wireless ad hoc networks. in INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE. 2003.

[34] Mahajan, R., et al., Experiences applying game theory to system design http://doi.acm.org/10.1145/1016527.1016531 in Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems 2004 ACM Press: Portland, Oregon, USA p. 183-190.
[35] D. Milojicic, F. Douglis, Y. Panedeine, R. Wheeler, and S. Zhou. Process migration. ACM Computing Surveys, 32(3), 2000.
[36] K. Dey, "Providing Architectural Support for Building Context-Aware Applications," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, USA, 2000.
[37] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D.dissertation, University of California, Irvine, USA, 2000
[38] D. Garlan, "Software architecture: a roadmap," in Proceedings of the Conference on the Future of Software Engineering, 2000, pp. 91-101.
[39] Sun Microsystems, "Jini Architecture Specification," December 2001. [Online]. Available: http://www.sun.com/software/jini/specs/jini1.2html/jini-title.html. [Accessed: Mar. 24, 2006]
[40] Mascolo et al., "Mobile Computing Middleware," Advanced Lectures on Networking : NETWORKING 2002 Tutorials, LNCS 2491, pp.20-58, Springer-Verlag Berlin/Heidelberg, 2002
[41] G. Cabri, L. Leonardi, and F. Zambonelli, "Mobile-Agent Coordination Models for Internet Applications," IEEE Computer, vol. 33, no. 2, pp. 82-89, February 2000
[42] Sun Microsystems, "Java Remote Method Invocation," 1997. [Online]. Available: http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html. [Accessed: Mar. 26, 2006]
[43] Sun Microsystems, "Java Message Service", April 2002. [Online]. Available: http://java.sun.com/products/jms/docs.html. [Accessed: Mar. 26, 2006]
[44] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," ACM Computing Surveys, vol. 35, no. 2, pp. 114-131, June 2003
[45] Gelernter, "Generative Communication in Linda," ACM Transactions on Programming Languages and Systems, vol. 7, no. 1, pp. 80-112, January 1985
[46] G. Cabri, L. Leonardi, and F. Zambonelli, "Engineering Mobile-agent Applications via Context-dependent Coordination," IEEE Transactions on Software Engineering, vol. 28, no. 11, pp. 1034-1051, November 2002
[47] Sun Microsystems, "JavaSpaces(TM) Service Specification," March 2006. [Online]. Available: http://java.sun.com/products/jini/2.0/doc/specs/html/js-title.html. [Accessed: Mar. 24, 2006]
[48] Johanson and A. Fox, "The Event Heap: A Coordination Infrastructure for Interactive Workspaces," in Proceedings of the 4th IEEE Workshop on Mobile Computer Systems and Applications, 2002, p. 83
[49] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D.R. Reddy, "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," ACM Computing Surveys, vol. 12, no. 2, pp. 213-253, June 1980
[50] F. Codd, "A relational model of data for large shared data banks," Communications of the ACM, vol. 13, no. 6, pp. 377-387, June 1970.
[51] S. Abiteboul, P. Buneman, and D. Suciu, Data on the Web: From Relations to Semistructured Data and XML. San Francisco: Morgan-Kaufmann, 1999
[52] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, Extensible Markup Language (XML) 1.0 (Third Edition), World Wide Web Consortium (W3C) recommendation, February 2004. [Online]. Available: http://www.w3c.org/TR/2004/Rec-xml-20040204/. [Accessed: Mar. 24, 2006]
[53] C. Fallside and P. Walmsley, XML Schema Part 0: Primer Second Edition, World Wide Web Consortium (W3C) recommendation, October 2004. [Online]. Available: http://www.w3c.org/TR/xmlschema-0/. [Accessed: Mar. 24, 2006]
[54] G. Klyne, J. J. Carroll, B. McBride, Resource Description Framework (RDF): Concepts and Abstract Syntax, World Wide Web Consortium (W3C) recommendation, February 2004. [Online]. Available: http://www.w3.org/TR/rdf-concepts/. [Accessed: Mar. 24, 2006]
[55] T. Berners-Lee, R. Fielding, and L. Masinter, Editors, Uniform Resource Identifier (URI): Generic Syntax, Network Working Group RFC 3986, January 2005. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3986.txt. [Accessed: Mar. 24, 2006]
[56] Paslaru Bontas, L. Nixon, and R. Tolksdorf, "A Conceptual Model for Semantic Web Spaces," AG Netzbasierte Informationssysteme, Freie Universität Berlin, Germany, Report B 05-14, 2005
[57] A. K. Dey, "Understanding and using Context," *Journal of Personal and Ubiquitous Computing*, Volume 5 (1), pp. 4-7, 2001.
[58] L. Barkhuus and A. Dey, "Is Context-Aware Computing Taking Control Away from the User? Three Levels of Interactivity Examined," in *Proceedings of UBICOMP 2003, 5th International Symposium on Ubiquitous Computing*, pp. 149-156. October 12-15, 2003.
[59] T. Strang and C. Linnhoff-Popien, "A Context Modeling Survey," *Workshop on Advanced Context Modelling, Reasoning and Management*, UbiComp 2004, September, 2004.
[60] H. Chen, T. Finin, and A. Joshi, "Semantic Web in the Context Broker Architecture," in *Proceedings of PerCom 2004*, Orlando FL., March, 2004.
[61] X. Wang, J. S. Dong, C. Chin, S. Hettiarachchi, and D. Zhang, "Semantic Space: An Infrastructure for Smart Spaces," *IEEE Pervasive Computing*, Volume 3(3), pp. 32-39, 2004.
[62] C. Intanagonwiwat, R. Govindan, D. Estrin, J. S. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 2-16, February, 2003.
[63] T. Buchholz, A. Küpper, and M. Schiffers, "Quality of Context: What It Is And Why We Need It," *Workshop of the HP OpenView University Association 2003 (HPOVUA2003)*, Geneva, July, 2003.
[64] D. Linner, I. Radusch, S. Steglich, and C. Jacob, "Loosely Coupled Service Provisioning in Dynamic Computing Environments," in *Proc. 1st International Conference on Communicatin and Networking in China (CHINACOM)*, October, 2006, to appear.
[65] C. Jacob, D. Linner, S. Steglich, and I. Radusch, "Autonomous Context Data Dissemination in Heterogeneous and Dynamic Environments," *IEEE Consumer Communications and Networking Conference (CCNC2007), Special Session on Autonomic Communications*, Harrah's Las Vegas, Nevada, USA, January, 2007, to appear.
[66] M. Mamei, F. Zambonelli, "Spreading Pheromones in Everyday Environments via RFID Technologies," *2nd IEEE Symposium on Swarm Intelligence*, June, 2005.

[67] R. T. Fielding, R. N. Taylor, "Principled design of the modern Web architecture," in *Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland, pp. 407-416, 2000.
[68] M. Baldauf, S. Dustdar, and F. Rosenberg. A Survey on Context Aware Systems. In *International Journal of Ad Hoc and Ubiquitous Computing*, 2006, forthcoming.
[69] *Sergey Brin and Lawrence Page (1998).* "The anatomy of a large-scale hypertextual Web search engine". Proceedings of the seventh international conference on World Wide Web 7, *107-117 (Section 2.1.1 Description of PageRank Calculation).*
[70] G. Canright, K. Engo-Monsen, and M. Jelasity, "'Efficient and robust fully distributed power method with an application to link analysis"," DELIS, Tech. Rep. 0320. [Online]. Available: http://delis.unipaderborn.de/paper/DELIS-TR-0320.pdf
[71] A.-L. Barabasi, "Linked: How Everything Is Connected to Everything Else and What It Means", 2000
[72] I. Clarke, O. Sandberg, B. Wiley, and T. Hong, ``Freenet: A distributed anonymous information storage and retrieval system," in *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, (Berkeley, CA), 2000.
[73] P. Druschel and A. Rowstron, ``Past: A large-scale, persistent peer-to-peer storage utility," in *Proceedings of the Eighth IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, (Schoss Elmau, Germany), May 2001.
[74] M. Waldman, A. Rubin, and L. Cranor, ``Publius: A robust, tamper-evident, censorship-resistant, web publishing system," in *Proceedings of the 9th USENIX Security Symposium*, August 2000.
[75] K. Amin, G. von Laszewski, and A. R. Mikler. Toward an Architecture for ad hoc Grids. In Proceedings of the International Conference on Advanced Computing and Communications, India, 2004
[76] T. Friese, M. Smith, and B. Freisleben. Hot Service Deployment in an Ad Hoc Grid Environment. In Proceedings of the 2nd Int. Conference on Service-Oriented Computing, pages 75–83, New York, USA, 2004.
[77] N. Medvidonic and R. Taylor, "A classification and comparison framework for Software Architecture Description Languages", IEEE Trans .on Software Engineering, Vol 26, num 1, 2000
[78] E. Bruneton, T. Coupaye, M. Leclerq, V. Quema and J.-B. Stefani, "The Fractal Component Model and its Support in Java", Software Practice and Experience, special issue on Experiences with Auto-adaptive and Reconfigurable Systems, vol 36, number 11-12, pp 1257-1284, 2006
[79] D. Caromel and L. Henrio, "A Theory of Distributed Objects: Asynchrony - Mobility - Groups - Components", Monograph, Springer Verlag, 2005
[80] L. Yamamoto (ed), Framework for distributed on-line evolution of protocols and services, BIONETS Project Deliverable 2.2.2, http://213.21.183.210/docs/BIONETS_D2_2_2.pdf
[81] BIONETS Deliverable D3.1.1, 2006
[82] BIONETS Deliverable D1.1.1, 2006
[83] D. Miorandi, L. Yamamoto and P. Dini, Service Evolution in Bio-Inspired Communication Systems. In Proc. of SOAS 2006, Erfurt, 2006
[84] D. Roman, U. Keller, H. Lausen, J. d. Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, D. Fensel, „The Web Service Modelling Ontology", IOS Press, vol.1, no.1, pp.77-106, 2005
[85] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, J. Su, S. Tabet. (2005, April). Semantic Web Services Ontology (SWSO). Available: http://www.daml.org/services/swsf/1.0/swso/
[86] The OWL Services Coalition. (2004, November).OWL-S: Semantic Markup for Web Services, Available: http://www.daml.org/services/owl-s/1.1/
[87] Q. Wall. Understanding the Service Lifecycle within a SOA: Run Time [Online]. Available: http://dev2dev.bea.com/pub/a/2006/11/soa-service-lifecycle-run.html
[88] U. Koivukoski, V. Räsänen. Managing mobile services: Technologies and Business Practices, John Wiley & Sons, March 2005.
[89] JAIN SLEE specification http://jainslee.org/slee/slee.html
[90] BPEL Designer http://active-endpoints.com/active-bpel-designer.htm
[91] BPEL for ECLIPSE http://www.eclipse.org/bpel/
[92] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D.dissertation, University of California, Irvine, USA, 2000