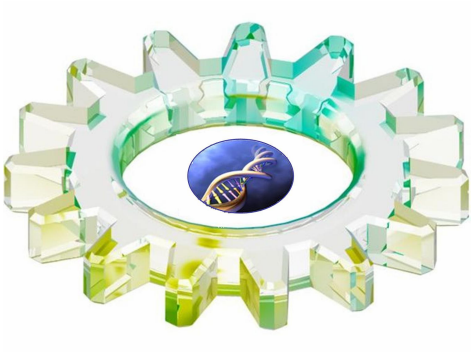

BIONETS

WP 3.1 – REQUIREMENTS ANALYSIS AND ARCHITECTURE

D3.1.2 – Refinement of Service Architecture and Requirements

Reference:	BIONETS/VTT/WP3.1/1.0
Category:	Deliverable
Editor:	Jyrki Huusko (VTT)
Author(s):	Jyrki Huusko (VTT), Janne Lahti (VTT), Francoise Baude (INRIA), Laura Ferrari (TI), David Linner (TUB), Heiko Pfeffer (TUB), Stephan Steglich (TUB)
Verification:	Paolo Dini (LSE), Daniele Miorandi (CN)
Date:	3.8.2007
Status:	Final
Availability:	Public

Executive Summary

The goal of this document is to provide the first refinement of the BIONETS service architecture and service requirements specification, based on the research activities carried out in work packages WP1.2, WP3.2. The main advances have been made in the areas of BIONETS service life-cycle. In this document, we clarify the impact of the life-cycle studies in WP3.2 to the overall service architecture. In addition, after the previous architecture deliverable [BIONETSD311], some advances have been made in the areas of communication and interaction framework for service architecture and application scenario analysis. In this deliverable, we concentrate on presenting the overall service architecture framework, including the main functionalities for the communication between services and underlying infrastructure. Also the concepts of Service Cells, Service Individuals, Mediators and Interaction framework are elaborated.

The application scenario analysis has been complemented in the Amendment of Deliverable D311, which was published in May 2007. In the amendment, four proposed application scenarios are analysed in more detail in terms of their evolutional and bio-inspired features. Thus, the revision of the application scenarios is not presented in the deliverable at hand.

The work on the integration of the BIONETS network and service architectures started in project month M19. This deliverable, in addition to deliverable [BIONETSD122], provides the initial framework for the integration of such components.

The preliminary security considerations for service architecture were handled briefly in the deliverable [BIONETSD311] and work on WP4 is currently ongoing to identify enhanced mechanism also for service architecture. In this document, we do not address any changes for the security requirements, which were previously introduced in [BIONETSD311].

Document History

Version History

Version	*Status*	*Date*	*Author(s)*
0.1	*Created*	*23.01.07*	*J.Huusko*
0.2	*Draft*	*19.05.07*	*J.Huusko*
0.3	*Draft*	*24.05.07*	*J.Huusko*
0.4	*Draft*	*07.06.07*	*F. Baude*
0.5	*Draft*	*20.06.07*	*L.Ferrari*
0.6	*Draft*	*21.06.07*	*D. Linner*
0.7	*Draft*	*04.07.07*	*L. Ferrari*
0.8	*Draft*	*04.07.07*	*D. Linner*
0.9	*Draft*	*09.07.07*	*J.Huusko*
0.10	*Draft*	*10.07.07*	*J.Huusko*
0.11	*Draft*	*11.07.07*	*F Baude*
0.12	*Draft*	*12.07.07*	*D.Linner*
0.13	*Draft*	*18.07.07*	*J.Huusko*
0.14	*Proposal*	*26.07.07*	*J.Huusko*
0.15	*Review*	*03.08.07*	*P. Dini*
0.16	*Review*	*31.07.07*	*D. Miorandi*
1.0	*Final*	*03.08.07*	*J.Huusko, J. Lahti, D. Linner*

Summary of Changes

Version	*Section(s)*	*Synopsis of Change*
0.1	*Not Applicable*	*Template and initial Table of Contents drawn*
0.2	*ToC*	*Some modifications to the deliverable structure*
0.3	*Table of contents*	*New table of contents according to S-A meeting*
0.4	*Changes in Sect. 2.3, 3.1, 3.2, 3.4*	*Added content to the respective sections*
0.5		*Based on INRIA first input + TI contrib.*
0.6	*Changes in Sect. 2.1 and 2.2.1*	*First draft for node-level autonomy, some text block for architecture overview*
0.7	*Section 2*	*TI's comments after VTT TUB mails*
0.8	*Added text to Section 3.3*	*TUB's contribution to section 3.3. added*

0.9	*Complemented Section 4, added text to Section 2*	*Added the communication framework and basic network interface description. Some modifications to Section 2.1*
0.10	*Executive summary and future work*	*Added the executive summary and future work sections.*
0.11	*Changes in 2.3, 3.1, 3.4*	*Revised 2.3 with additional state of the art reference. Reshaped the introduction of 3.1.1, conclusion of 3.1.2. and incorporated TI's comment of 3.4*
0.12	*Changes in sections 2 and 3*	*Complemented sections 2,3*
0.13	*Changes in section 1,2,3, 4 and 5*	*Complemented sections 1,2,4 and 5, small correction t the other sections, references added*
0.14	*Changes in all the sections*	*Complemented the sections and some errors corrected. Distributed for internal review.*
0.15	*Changes in all the sections*	*Internal Review by Paolo Dini*
0.16	*Changes in all the sections*	*Internal Review by Daniele Miorandi*
1.0	*Changes in all the sections*	*Final editing based on the review comments*

Note

Reviews after final document delivery (Version 1.0) to the project may or may not result in modifications to the document. If modifications post review is necessary, then the first version of the resultant document is 1.1.

Contents

1. Introduction

One of the goals of the BIONETS project is to build a versatile service architecture for a network environment which is “disappearing” and “disconnected”. In practise, this means that the physical network transmission is more or less invisible to services operated by U-nodes and the services and data provided by T-nodes is accessible to devices only within the island of connected nodes. The “disconnected” nature of the network implies that the connectivity of the nodes is not guaranteed, and that it depends on the time and situation, i.e., environment, user interaction and system resources can affect to the connectivity. For these reasons, the service architecture needs to be also highly versatile, robust, secure and dynamic. In [BIONETSD311], we described the initial architecture principles for the BIONETS service architecture, including the layered structure of operations for the T- and U-nodes, considerations about security issues, service framework, service generation, autonomicity and evolution. In addition, we defined several application scenarios, which gave the basis for defining the architectural requirements.

In this deliverable, we present a revision of the architecture description, based on the work carried out during the last months. The purpose of this deliverable is to assemble the main results and provide enhanced insight to the architecture, which can be further utilized in the upcoming research activities. The application scenarios were revised for the amendment of [BIONETSD311], and thus the revision is not included in this document. The document is arranged as follows:

In Section 2, we describe the overall service architecture framework, which supports several new functionalities. These new functionalities include: the Service Mediators, Interaction Framework, and Service Cells/Service Individuals. The Service Framework defines also the logical interfaces between these different components and the underlying network infrastructure. We also introduce a distinction between the Service Cells and Individuals as well as between node- and service-level autonomicity. In Section 2.4, we briefly discuss the service architecture evolution. The current BIONETS system concentrates mainly on the evolution of services. However, in order to provide a versatile infrastructure, also the evolution of the architecture needs to considered. One simple example of architectural evolution could be the transition from node-level autonomicity to purely service-level autonomicity, or long-term adaptation of interaction framework interaction models and mediator functionalities.

In Section 3, we discuss the service life-cycle support. This was studied in WP3.2 for the service architecture, including the service provisioning and usage considerations as well as the semantic reasoning, service adaptation driven by Service Mediators, self-aggregation, and evolution of Service Cells and Individuals. From the evolution and adaptation point of view, we are distinguishing between the short-term adaptation and long-term adaptation. From the service and application point of view, the short-term adaptability refers to the ability of reacting just-in-time to changes in the physical configuration of the environment, such as network configuration, service availability or adaptation to rapid changes in channel and network conditions, aiming to the realization of robustness and QoS in the system. The long-term adaptability refers to “learning” of alternative approaches in order to handle a particular user request (e.g. ability to exchange services, modification of service implementations, reconfiguration of parameters), aiming to progressively increase the overall adequacy of the system response by the use of evolutionary principles.

In Section 4, we introduce a slightly more detailed communication framework for the Service Framework, in order to lay the foundations for the network and service architecture integration. In this section, we give some simple examples of data and control messaging based on the generic BIONETS messaging and addressing [BIONETSD112]. The communication framework and the definition of the required control and data messages for service architecture are addressed in the network and service architecture task force during the WP3.1 Task 3.1.3.2. Finally, in the Section 5 we address some additional future work, which is in progress and which will be integrated in the service architecture description and principles during the next working period.

2. Refinement of the service architecture

2.1 Service framework overview

In this section, we briefly describe the advances in service framework achieved after the submission of [BIONETSD311]. Figure 1 illustrates the current overview of the service framework and the elements provided by T- and U-nodes. The framework can be divided coarsely into three parts. The upper part includes the application (i.e. service cells and individuals [BIONETSD321]) and Management, the second part includes the Interaction framework and Interaction footprint and the third part the actual BIONETS network interface.

The BIONETS service architecture is built on the notion of abstracting physical and virtual devices by nodes, while we distinguish between U-Nodes and T-Nodes as depicted in Figure 1. In the centre of the service architecture there are two complementary architectural elements, Services and Service Mediators. While the Services realize application functionality, Service Mediators implement all environment-related organization tasks. For instance, this also includes the service life-cycle management.

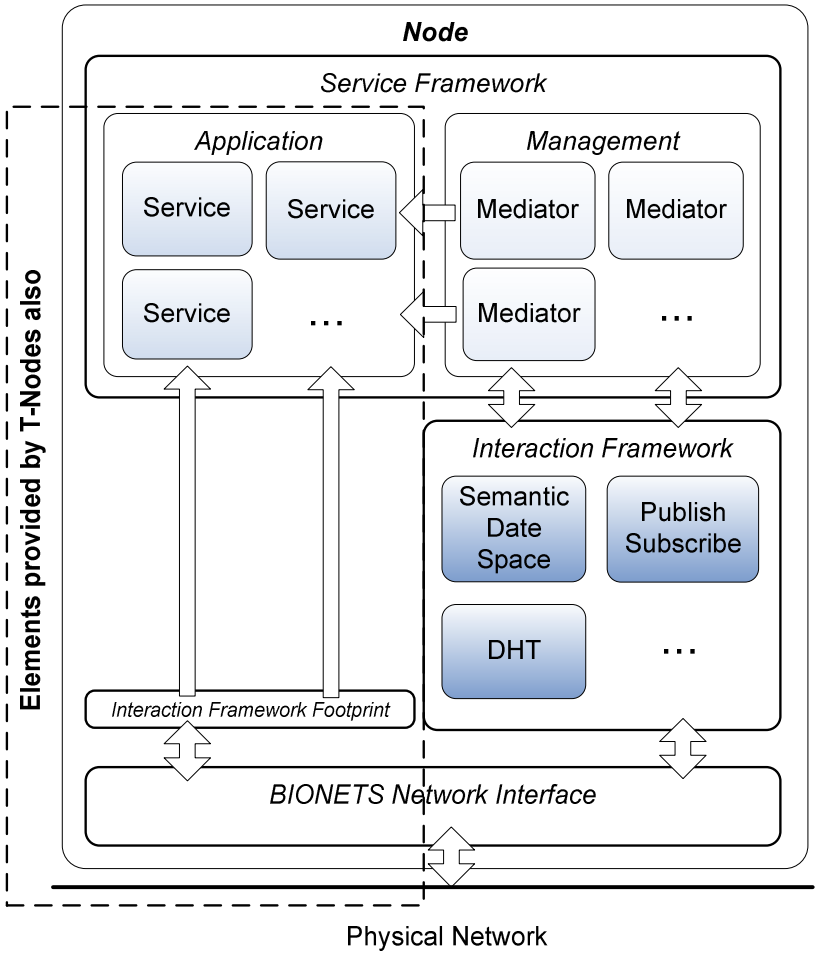

Figure 1: Overview of U- and T-Node architecture

T-Nodes are in D1.1.1/2 characterized as simple devices with very limited resources such as CPU performance, storage capacity, etc. Accordingly, we do not suppose them to actively participate in cooperative management task, but rather to be more or less managed externally. Nonetheless T-Nodes provide information, e.g. sensor data. Corresponding to the principles described in [BIONETSD321] (service execution is a state change) data provision is one facet of services, so the data-provision capabilities of T-Nodes can be modelled as a service, which significantly facilitates the integration into the service framework. In this regard, simple data processing tasks could also be handled by T-Nodes (e.g. transformation of a geographical position from degree, arc minutes and arc second into a decimal representation), as claimed in Figure 2. However, allowing services on T-Nodes implies that management/control functionality can only be integrated with T-node services if an externalized execution environment for self-management logic is available, due to restrictions of T-nodes and limited processing power [BIONETSD111].

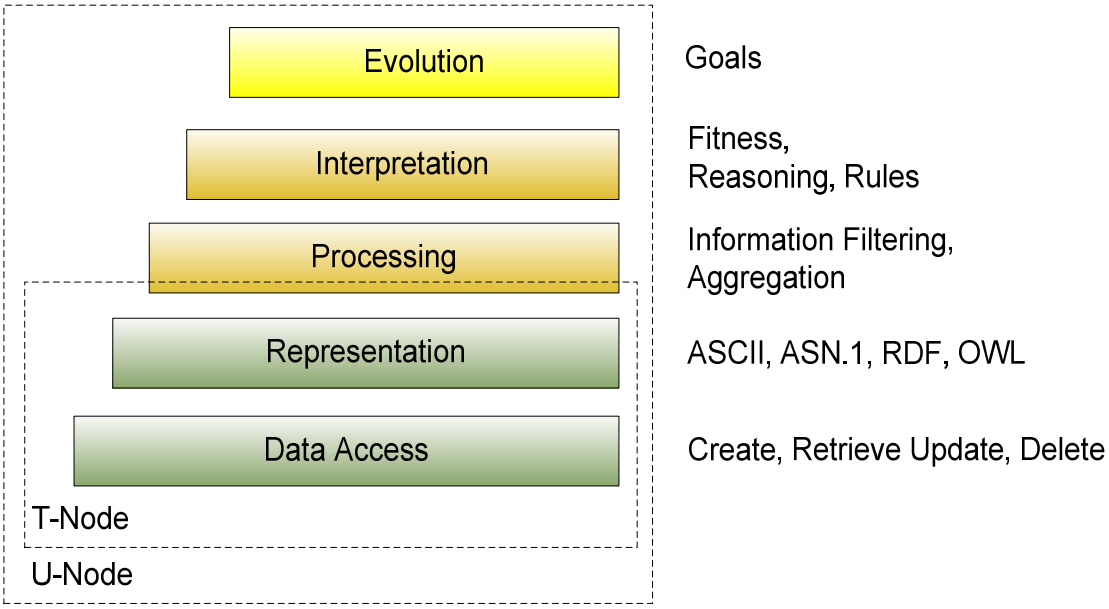

Figure 2: Layered view operation complexity

The notion of Interaction Framework was already introduced in [BIONETSD321]. The Interaction Framework can be seen as a container for communication middleware systems, which implement predefined patterns, the so-called interaction models. Following the layered view outlined in Figure 1, the Interaction Framework is only available to Service Mediators and consequently not included in the service architecture subset for T-Nodes. Instead, an Interaction Framework footprint is available to support the accessibility to services through the BIONETS network interface with CRUD (Create, Retrieve, Update and Delete.) principles. This footprint is, as an interaction model, a means to support distributed but cooperative self-organization algorithms implemented by the mediators. One of these interaction models is the Semantic Data Space, which was already introduced in [BIONETSD321].

More generally, the Interaction Framework provides a shared data space for content delivery, service control, message management, and semantic data delivery over the physical network. The purpose of the Interaction Framework is to provide flexibility in terms of having multiple concurrent and evolvable interaction model implementations. In order to implement the shared data space for the U-node, several interaction

models can be considered, e.g. in the serverless manner with the distributed hash tables, or with asynchronous publish/subscribe messaging, or the Semantic Data Space. The actual implementation of the shared data space needs verification *per se*, taking into consideration different need of different message types and data delivery, using for example scalability as a metrics when deciding the best suitable interaction model. These issues will be taking into consideration in the upcoming deliverables for WP3.2 and WP1.2 (BIONETS networking primitives). In the section 2.1.3 of this document, we introduce the Semantic Data Space as one of the possible interaction models for the Interaction Framework.

2.1.1 Service Cell

The desired features of a Service Cell should include the ability to dynamically change and adapt its internal behaviour according to modifications in the environment, to interact with other Service Cells, aggregating and cooperating to provide high-level services, and to make use of autonomic reasoning in order to enable self-* capabilities both at the single unit level and at the overall service-level. Based on this, a Service Cell should be characterised, at least, by [KCLOW05] [CASCADAS31]:

- A description of the services: a representation of the (semantic) information regarding input/output effects and pre-conditions. The effects represent the global objectives of the service;
- Non-functional characteristics, e.g. policy, QoS. They can be expressed with semantics as well;
- A model of the state: this can be seen as a workflow, constituted of states and transitions between states, which models the Service Cell behaviour;
- And a rule-based model for describing and restricting its behaviour. The rules are executed whenever a change in the internal state of the component or in the environment triggers them. It can work on the semantic representation of the service cell (i.e. effect, input/output,…) or on the state of the service cell. One of the tasks for the Service Mediator is to provide this kind of reasoning engine functionalities.

The state model describes the possible internal states for a Service Cell and the possible transitions between pairs of states. In other words, it could be defined as a state machine; then the state model is a description of the steps the Service Cell will execute to achieve its internal objectives (i.e. the behaviour of the service cell).

The transition functions are the specific functions: they indicate a state change and are described by an event, a condition that would need to be fulfilled to enable the transition, and then the specific function to be executed.

The Service Mediator as a reasoning engine (which can be an inference algorithm, or a rule-based system) executes the state model, and its main role is to keep track of:

- The state reached in the state model execution. Eventually, it may take trace of the history, storing the previous states of the state model.
- The environment: any request coming from other service cells.

Mainly, it is able to run the state machine used to describe the state model. It checks if a transition may take place, runs the transition, invoking the proper specific features

-if specified- and properly represents the semantic description of the new reached state.
The reasoning engine is the core of the autonomic part of the service cell, adapting its behaviour to the changed conditions, situations, fault etc… The reasoning engine can also deal with semantic information (input, output, effects) in order to interact with the environment, in case the Service Cell provides a semantic description. This kind of semantic description can be used for discovery or for service composition. Non-functional properties can be used for evaluating the fitness.

2.1.2 Service Individual

In D3.2.1 the Service Individual was introduced as the second basic element for the realization of application logic in the BIONETS service framework. A Service Individual is a blueprint for a composition of Service Cells aiming at the provision of high-level functionality of potential interest (e.g. directly requested) for the user. In contrast to a Service Cell, the code base of Service Individual is intended to be described in an open manner, based on a simple scheme. Currently the scheme is planned to comprise:

- A set of placeholders for Service Cells matching a particular description.
- A Parameter set assigned to each placeholder, which will be applied for the configuration of the Service Cell substituting the respective placeholder later on.
- Several graphs (composition plans), for instance determining the flow of information among the Service Cells, the temporal execution order, or the flow of fitness information.
- Optionally, control expressions (if, sequence, parallel, etc.) for constructing the graphs.

The definition of this scheme is part of on-going activities in WP3.2. The purpose for this graph-oriented way of realizing Service Individuals can be found in two important characteristics. Firstly, Reusing existing application elements (Service Cells) is a straightforward but powerful and proven approach to generate more value for the system. Secondly, the stringent code base representation provides an innovative field of application for evolutionary transformation by genetic operators. Since Service Individuals do only refer to other Services in a generic way (just placeholders with descriptions) they can be regarded as interpretable knowledge representations. In this regard they are closed entities which may be deployed across different nodes and islands, while their interpretation always depends on the current system context (e.g. actually available services to substitute the placeholders).

The kind of interface for utilizing Service Individuals is equivalent to the one of Service Cells. The same holds for the semantic descriptions. Only the interpretation of Service Individuals (execution and runtime control) is a bit different, as Service Individuals will not need support for native operation on the platform hosting the BIONETS node. Further details about the provision and usage of Service Individuals are addressed in the sections 3.1 and 3.2.

2.1.3 Interaction Framework

In the deliverable [BIONETSD321] we have presented the Semantic Data Space solution for the BIONETS Interaction Framework. In more generic way, the Interaction Framework can be specified as a shared data space as described in previous sections of this deliverable. The different proposed solutions such as the

Semantic Data Space and Distributed Hash Tables for the Interaction Framework have an defined impact on the overlay network that will be defined for BIONETS. Thus the only concern is not analyze only the Interaction Framework from service point of view and find the proper solutions for each service oriented purposes but also to find the proper interfaces satisfying different implementations of Interaction framework for overlay network side. In the following sections we concentrate on Semantic Data Space functionalities, which are currently under the studies.

The information that should be part of the Interaction Framework should cover the functionalities of interactions provided by services. There are different modalities of working, e.g. notification, one-way, solicit-response, request-response, etc. A service has input and output messages depending on the mode. The message contains one or more parameters that have data type (that can be primitive or derived). While data types are important for the automatic matching of message parameters, they do not capture the semantics of those parameters. The *business role* is then introduced for the service in order to give semantics to the parameters (it takes its value from a pre-defined taxonomy for *business roles*). Each service can be semantically described through its purpose and category. The purpose represents the business functionality provided by the operation (the business role can be expressed through a taxonomy). The category of a service represents the domain or area of interest for the current service.

Several services may provide 'similar' objectives in terms of their mode, message, purpose and category. It is thus important to define qualitative properties for services: fee, security, privacy, etc.

In order to define a service the following features have to be described:

- The mode
- Input and Output message
- The purpose (i.e. objective, effect, goal)
- The category
- And the quality.

It is necessary to have a set of semantic meta-data formally expressed in order to have a multi-dimensional description of a service. A specific vocabulary (an ontology) should exist in a specific domain. The Semantic Web provides tools and an approach for the semantic representation of services. Formalisms like OWL and OWL-S allow the possibility to create ontologies, and there are engines that can work on them.

More specifically in order to describe a service the following characteristics are necessary: Input, Output, Pre-conditions and Effects (IOPE). The semantic annotations of input and output are used in the algorithms that manage the composition of services; the semantic annotations related to the effect are used in the research of services that can satisfy the request, the semantic annotations for pre-condition represent links to the service state that have to be satisfied before invoking the operation

The Interaction Framework provides the means for service composition and interactions. The composability model for service can be based on composability rules to compare syntactic and semantic properties of services. Syntactic rules include 1) mode composability, which compares operation modes, and 2) binding composability, which compares the binding protocol of the interacting services. Semantic rules include 1) message composability, which compares the number of

message parameters, their data types, business roles, and units; 2) semantics composability which compares the semantics of the services, 3) qualitative composability, which compares qualitative properties of services; and 4) composition soundness, which checks whether combining services in a specific way is worthwhile.

The Semantic Data Space introduced in [BIONETSD321] plays the role to host the semantic concepts introduced, depending on the purpose and requirements covered. The Mediator plays the role of managing the exchange of semantic information in the SDS using the create, read, update and delete (CRUD) operations. This information will be used as well by a reasoner or semantic reasoner part of the Service Cell/Individual (or Mediator in this phase). The Semantic Data Space should allow information search and knowledge acquisition/extension of the distributed semantic of the service cell, according to pre-defined query models and ontology matching techniques. The distributed ontology represents a network of concepts, where each concept is characterized by a set of attributes and a set of relationships with other concepts exchanged trough the Interaction Framework.

In BIONETS the service cell/individual act as autonomous and independent components and share knowledge by submitting discovery queries and by replying with relevant knowledge. To this end, ontologies are generally employed for describing the knowledge to be shared, and appropriate techniques for consensus negotiation are required to deal with the different concept meanings in the ontologies provided by different components (that represent different semantic islands of information that need to interact and exchange information).

A novel approach to the self-organization of autonomous islands of components is based on some semantic handshake techniques to handle the problem of consensus negotiation and agreement to commit a declaration of interest to a semantic island, by matching ontologies for organizing the committed components according to a structured organization for efficient query propagation In the automatic generation of composite services, self-aggregation and self-adaptation semantics can be a support in order to realize automatic/autonomic generation of composite services. The composition model should be based on syntactic and semantic features.

In Figure 1, Publish/Subscribe and DHT are meant as further examples and possible approaches to the Shared Data Space, and not as definitive elements.

2.2 Autonomic functionalities

Autonomicity can be interpreted as a combination of several self-* characteristics, primarily focusing on self-optimization, self-improvement, and self-adaptation (self-recovery, self-healing). Self-optimization and self-improvement address the different levels of evolution we are going to address within the service framework. On the one hand parameter sets for the configuration of services are permanently adjusted, on the other hand structural changes to service are utilized. The structural changes lastingly modify the appearance of a service from the system's point of view to create new, but semantically equivalent, services from existing services as explained more in detail in Section 3.3.1 of this document. Both self-optimization and self-improvement imply self-monitoring and -evaluation mechanisms. In contrast, self-adaptation addresses the system capability to respond quickly to changing environment conditions (e.g. disappearing nodes and implicitly services, varying link quality).

In a distributed computing environment most of these features need to be realized as distributed, cooperative models and algorithms to have a significant impact on the system behaviour visible to the user. The service model, i.e. the way a service is integrated in and interfaced by the system is the same for all BIONETS services. Accordingly, (self-) organization requirements, which do not depend on the purpose (in terms of the application functionality) of the service, will likely be equal for all services.

In the BIONETS system we address two different levels of autonomicity: node-level autonomicity and service-level autonomicity. The division for the node-level and service-level autonomicity originates purely from practical reasons. In some cases, it can be shown that the node-level autonomicity performs better in terms of complexity and scalability, although service-level autonomicity can provide higher flexibility to the system.

The proposed node-level autonomicity is a first step towards the actual service-level autonomicity and service-centric self-management and control. For service autonomicity, the Service Mediator can be seen also as a tightly coupled part of either the Service Cell (or group of Service Cells) and the Service Individual. By providing generic evolvable interfaces for Service Mediators and Services, as well as generic interfaces for the Mediator and Interaction frameworks, the system can evolve from pure node-level autonomicity to support also the full service-level autonomicity. As an example case of service migration, if the Mediator, which is controlling the functionalities of single Service Cell, has a generic interface for the Interaction framework, it is possible to migrate both Service Cell and Mediator without breaking the upper layer Mediator/Service Cell interface and possibly also without changing the internal functionality of the Mediator in the target node.

2.2.1 Node-level autonomicity

In order to avoid duplication of self-organization logic, in terms of storage-eating program code and processes consuming runtime-resources (e.g. CPU, memory), the Service Mediator is introduced in the service architecture. The Service Mediators support service autonomy, without being integrated with the services themselves. Service Mediators are autonomous applications mirrored on all nodes that need or wish to support a particular organization feature, as illustrated in Figure 3. For that purpose they may also be duplicated and distributed among nodes. Service Mediators are supposed to implement all aspects of autonomy that are not application-related and therewith form the dynamic glue logic holding the services together. For example a migration mediator mirrored on two nodes may recognize the need to move a service from one node to the other in order to guarantee the seamless provision of this service to the user. Even the algorithms handling the phases of the evolutionary service life-cycle, which is outlined in [BIONETSD321], are supposed to be implemented in Service Mediators. This is explained more in detail in Section 3.

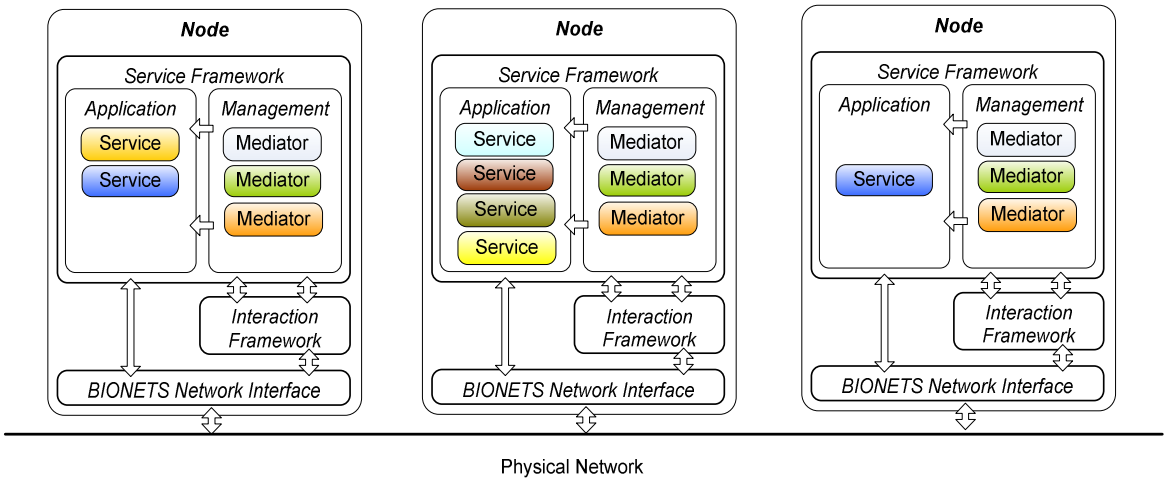

Figure 3: Services vs. Service Mediators

2.2.2 Service-level autonomicity

In the node-level autonomicity, the logic providing all the autonomic functions needed in the service is located in the node hosting the services. This approach, as explained in the previous chapter, brings many benefits for the BIONETS Service architecture. For example, the Services can be designed to be more light-weight and the code duplication of cooperation-related processes on one node is minimized. Another benefit is that the services are kept simple; therefore, they may also be provided by T-Nodes and not just U-Nodes. Also security may benefit from the decoupling of framework-related tasks and the functions provided by Services. The downside of this approach would be the classic appearance of services in the BIONETS service architecture. The services would not be autonomous themselves and thus the essence of BIONETS, where services act like biological entities without centralized management block, would not be realized.

Incorporating autonomy at the 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 at the same time it would result in the duplication of functionality and resource consumption (overhead due to the management actions integrated in each Service Cell). Integrating all the self-organization and self-management functionalities to the Services would require a lot of management logic to be implemented in the Service in addition to the actual application logic, which is Service-specific. This approach would require making each service partially intelligent. This would give some more flexibility for the system to cope e.g. with the service mobility, although increasing the complexity and control signaling.

The essence of BIONETS is the evolution of BIONETS Services, where each specific Service evolves through generations and may end up offering better services. If the Service logic which implements the self-organization and self-management functionalities, like e.g. migration, were integrated in the Service itself, then it could be possible that also all the autonomic functionalities. e.g. migration would go through the evolution phase, thus providing better and more efficient autonomic functionalities.

In the following Figure 4 we can see an example of autonomic functionality of the BIONETS Service Cell. Figure 4 illustrates two different cases of service migration.

Case 1: *Basic migration of Service Cell* illustrates perhaps the simplest form of Service mobility in the BIONETS environment, where one Service Cell entity migrates from U-Node A to U-Node B. The trigger for the migration process could be for example a situation where Service Cell X was offering some service to the user in the U-Node A and the user switched to U-Node B. In this case we can analyze where the 'autonomicity' or the actual decision-making logic for the Service migration should be located.

In Case 1 the Service is providing a service to the user using Node A and when the user switches to Node B the Service Cell X needs to follow the user in order to offer the best possible service quality to the user. In this case the most natural place for the decision making logic is the Service Cell itself. The Service Cell X knows who it is serving and where its users are. Also when the Service Cell migrates to the destination node all the management logic is transferred along with the application logic. This of course increases the complexity of the migration process and also the amount of data traffic and control signalling, but also makes sure that all the necessary autonomic functionalities are available in the host node.

In Case 2: *Replication and migration* the actual migration process and related technical challenges are very similar to Case 1. What is different is the motivation of the process. In Case 2 the Service Cell X01 is providing very valuable service X to other services in the vicinity when it detects e.g. low resource availability in its host U-Node. Based on this knowledge it decides to replicate itself producing Service Cell X02 and migrating it to a new node guaranteeing the continuous availability of Service X.

In Case 2 the decision maker could be the Service Cell, or it could also be the node that is hosting the service. For example, the Service Cell X01 could be a very light weight Service Cell, which doesn't contain the management logic needed for the replication and migration. In this case the node could be the entity which triggers the replication and migration process and also provides all the needed functionalities and management logic for the whole process.

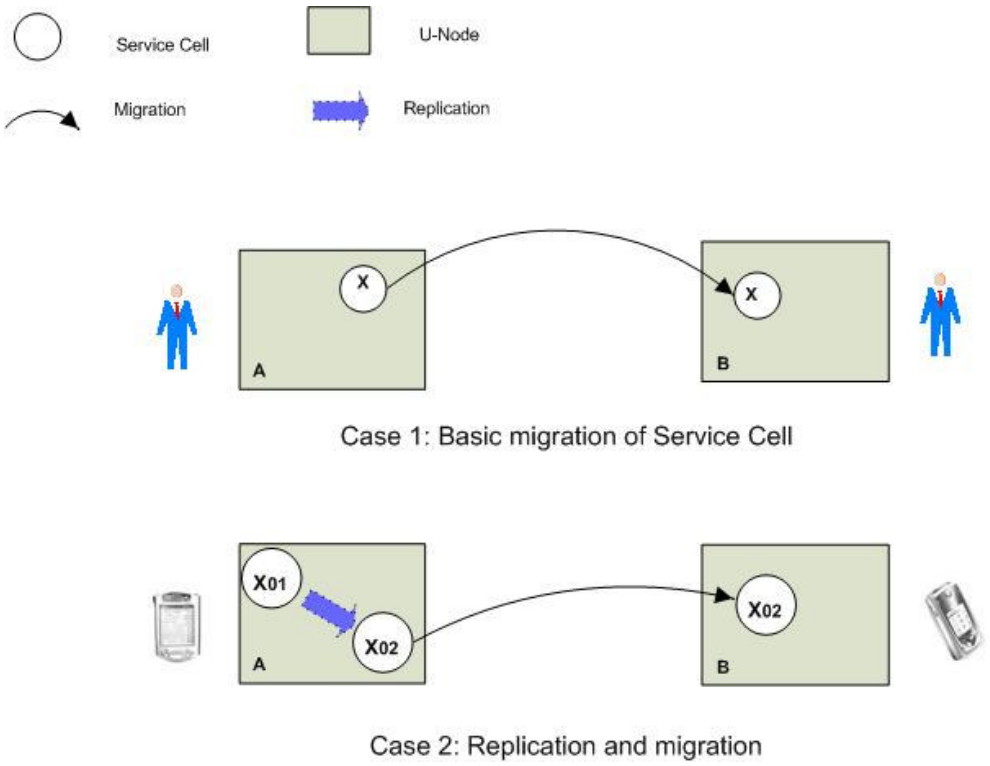

Figure 4: Two cases of Service Migration process

2.3 Evolution of service architecture

The focus of the research conducted w.r.t. bio-inspired networked services is more on the evolution of services themselves. Nevertheless, it might also require to base these services on a service support that is also capable of dynamic changes, be they adaptation or evolution. For this, we require that the adopted architecture be sufficiently open, so that the service support/framework could also be changed on the fly, even if already deployed on T-nodes and U-nodes. For instance, the service lookup protocol may be adapted [PVE05] in order to improve its performance and efficiency (how fit it is to discover services according to the service requests that include semantic and type information, etc).

To this aim, we suggest that all the infrastructure to host and manage services be itself conceived as a dynamically reconfigurable software. The BIONETS service framework implementation could get some inspiration of works dedicated to build adaptable middleware [GBS03, HLD04, RM06]

3. Service life-cycle support in architecture

The following Figure 5 illustrates the service life-cycle that was proposed in D3.2.1 [BIONETSD321], the self* list of properties of the autonomic computing should influence each step of the service life-cycle in order to realize a bio-inspired service life-cycle.

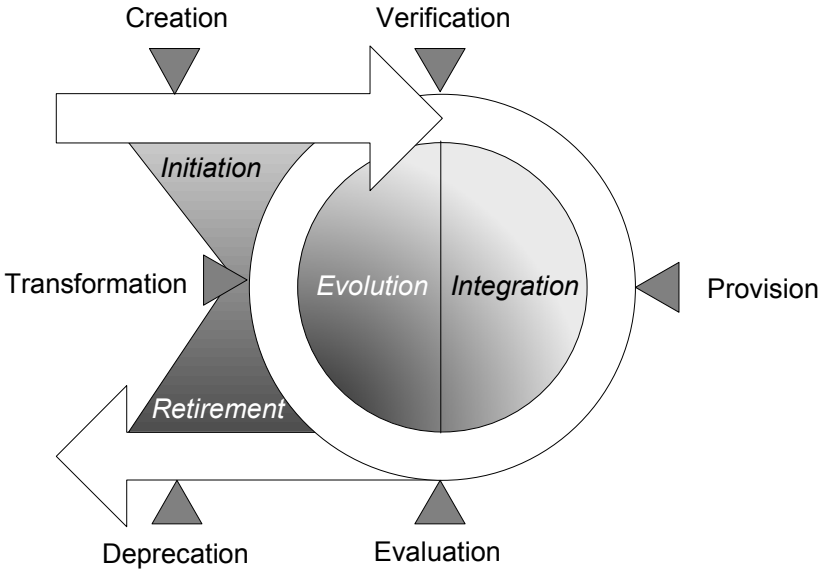

Figure 5: An illustration of Service life-cycle

In this service lifecycle the following phases and aspects will be underlined:

- Service provisioning
- Service usage
- Service adaptation
- Service evolution

3.1 Service provision

Service provision marks the point in the service life-cycle where newly created or transformed Service Individuals and Service Cells are made available for usage for the first time. For both kinds of emerging services the required, prior step is the creation of a service specification according to a user-delivered request. The service provision itself is comprised of the Service Deployment and the Service Announcement.

Service Deployment addresses the distribution of Services and Service Cells on the targeted set of nodes (possibly, on only one node) by means of cloning and migrating the clones. Even already running services from one set of nodes to a new one (with associated code loading so that the migrated instance has local access to the code it runs) may be part of the Service Deployment, for instance to pro-actively guarantee that a Service Individual can be instantiated later on. The purpose of service

announcement is to make all entities aware of the availability of a new service. For the moment we aim to investigate three potential ways for the realization of Service Announcement. All ways assume an initial node-wide announcement of services on their arrival and Discovery Mediators on each node that take care of BIONETS-wide service discovery.

1. The service Discovery Mediator starts brokering the services. Therefore, the Service Mediator waits for requests sent throughout the network and matching the description of their brokered services. If such a request arrives, the execution control by another Service Mediator is triggered indirectly (e.g. be posting a respective instruction to SDS).
2. Services (Service Individuals and Service Cells) are periodically announced island-wide by the Discovery Mediators. Each announcement includes a reference to address the service instance later on, the semantic description of the service (at least the part required for matching), and non-functional information, like parameters pertaining to the offered quality of service as resources type and amount, price to pay. Discovery Mediators on other nodes may obtain these announcements and evaluate them when a service request is raised on their respective node.
3. Services are actively searched by Discovery Mediators on purpose. The search message contains the description the desired service should match, again including non-functional properties. The response to such discovery request does then contain a reference to the service instance again.

For the service provisioning several subjects are still under research, for example in the following directions within the given BIONETS constraints. The services are not publishing themselves in some conventional – even if distributed – registry; on the contrary, service individuals regularly announce their presence in their vicinity, in an autonomous and totally decentralised (peer-to-peer) manner, which requires appropriate service locationing techniques [SBD07]. Consequently, it is not clear it we need to explicitly distinguish the publication of service types in some distributed semantic space, from the announcement of service individuals offering such services types: when a service individual announces its presence, it also announces the service that it offers, in semantics terms but also in more protocol-oriented terms, like, how it can be called. But, it may happen that both mechanisms are needed: for instance, if a service description is stored in the registry, it can be used as a form from which a new individual can be instantiated.

In the service provisioning phase the semantics can support the self-aggregation: semantics can help in service composition and service creation (referring to Figure 5, it can help in the phases A, B, C1). Service creation can be semantically guided, through effect (and with the use of input/output). Interestingly this process can be reused for evolving the service composition (i.e., it can help in the phases E, C1, C2 along the autonomous adjustment dimension, see Figure 5).

3.2 Service usage

Service usage pertains to the process of first discovering a service individual that can fulfil the need; second, the capability to really have this service individual serve a specific request and provide an associated reply to the requester. Due to the disconnected and volatile nature of BIONETS, we think that it is more relevant not to try to separate these two steps [LRSJ06]:

1. a service request is posted and routed on the BIONETS, considered as an interaction medium, possibly forwarded in more than one copy. The objective is to match the requested service with the services that have announced themselves (semantic, typing, available typing conversions must be considered in this matching phase). Request routing could be implemented as an addressless but content-based multicast operation, which fits well with the idea that this request routing indeed corresponds to the service discovery phase; moreover, the multicast routing could get inspiration of evolvable epidemic routing protocols [ACMN07], in order to evolve to a routing scheme that tends to maximise service hits while minimising the routing effort (i.e. consumption of routing resources).
2. eventually a service request will hit some nodes that are offering the requested service, and that autonomously may decide to serve the request or not (the decision might be taken according to some possible associated reward, and on the contrary, the decision to not serve may induce a possible associated punishment. This depends on the strategy for service selection and invocation, e.g. inspired by game theory or by economic or social schemas). Once served, the reply will be addressed (in a point-to-point manner) back to the service individual that requested the service execution
3. once a reply is received, the requester will evaluate the fitness of the reply, and indirectly, the fitness of the individual that produced this reply. To this end, the user might be explicitly asked to give a feedback. Although the system provides means for automated evaluation of functional and non-functional components [BIONETSD322], the system needs to provide also possibility for user interaction. Depending on the service invocation strategy we choose, it may happen that an additional reward or punishment may be sent to the individual that produced the evaluated reply. We can also envisage that if the reply evaluation is considered not good enough, the requester may wait for a given time period in case other replies to the initial service request arrive; or, it may decide to resend the same or slightly modified request. It may also happen that according to the result of the evaluation, the service individual that posted the request modifies itself (e.g., it changes its underlying organisation, so that next time, it will not anymore request such a service, or without the exactly same parameters or constraints): this is more pertaining to service adaptation.

It must be noticed that in order to serve the request, a selected individual may itself trigger some other service requests, and wait for associated replies before it can produce its own: service usage is a recursive process. Consequently, service usage payment (if any), evaluation and modification must also be designed as recursive processes.

Service usage payment seems to be especially adequate in our case of recursive service usage: the price for the usage of a given service can be established as the (weighted) sum of the prices of the other service individuals that must be requested in order to fulfil the service [JM06]. The network effort in order to support service usage could also be paid. For instance, we could consider that a fixed percentage –10% -- of the amount to be spend to have a request be fulfilled is dedicated to the underlying network that routes requests and replies. Also, the network actor –acting here both as a service discovery engine and as a more classic messaging support -- may be motivated to perform in an efficient manner, as itself may have to distribute the income according to the network nodes that contributed to fulfil the service usage: the fewer nodes may have been involved, the better the income for each of them. This same kind of motivation may apply at the level of service individuals,

which may consequently adapt or evolve: the objective is to improve their efficiency, expressed as how well they are evaluated regarding how much they need to spend themselves to do so.

3.3 Service adaptation

Service adaptation refers to the ability of the system to optimally configure itself reacting to the current conditions in the computing environment it is operated. In general, we distinguish between two methods of adaptation. The first method is based on the assumption that each Service Cell is delivered with an interface that allows the modification of properties for its containment. This interface could be realized by setting of name-value pairs, which describe the restrictions of the outer environment for the service. For instance, parameters could define the maximally allocable memory for the service execution or tell the service which kind of process scheduling is used by the host system. These containment parameters are supposed to be distinct from the configuration parameters that explicitly define the service behavior and are modified during the service transformation. When considering semantic service description according to the IOPE scheme, also semantic description for the containment parameters should be added.

Since service adaptation is always considered in conjunction with service execution the second method of adaption addresses the handling of service instances. Following the notion of SOA, a service can be classified and addressed by an implementation-independent description of functional and non-functional characteristics and services interfaces. If two service instances, i.e. two components K and L, match the same description φ, they can be considered semantically equal. Hence, when requesting the execution of a service with a profile matching φ, either K or L can be selected. This opens up several opportunities to work with the instances in the context of service adaptation. 1) When the originally selected service instance K disappears from the service environment (e.g. caused by the disconnection of the hosting node) during service execution, the execution could be recovered by selecting L to continue. 2) When the hosting node of L risks to run out of resources (e.g. battery power), the execution of L could be paused and L including the current execution state moved to another node for resuming the execution. 3) A clone L' of L could be proactively created on another node in the current island of nodes, if L is frequently requested but the last service instance in the island matching φ.

Both methods of adapting services at runtime can be utilized in a reactive and pro-active manner. For example, an adaptation can be carried out when a misbehaviour was detected or in advance, when a misbehaviour is likely to happen. While the first case could be built on predefined error conditions, the latter case requires monitorable and continuous values that indicate a development over time, such as the charge state of a battery. In practice reactive and pro-active adaptation should be combined, since we cannot assume to always have enough knowledge for the anticipation of unwanted behaviour.

Knowledge about the system context, i.e. current system state and the development of this state over time, is the basis for efficient adaptation. In this regard, the actual adaptation process is comprised of monitoring the system context, interpreting the gathered information, deciding on appropriate adaptation methods, and executing these decisions. In the current Service Architecture these processes are realized by a special Service Mediator, referred to as Adaptation Mediator. Description and interpretation of context data can at the beginning be based on a closed ontology and a finite set of rules for the definition of unwanted conditions. For the provision of

context data, the Adaptation Mediator additionally performs the three roles Context Producer, Context Relay, and Context Consumer, as explained in [BIONETSD321]. Therefore, the provision of context data is based on appropriate interfaces of service execution environment and network adapter giving access to information about the current situation of local resources and state of the network. For example, The following Figure 6 illustrates the steps of the adaption procedure as explained in the following:

(1) Provision of data describing the current context of the node and neighbouring nodes if available (e.g. load, available bandwidth, battery power, mobility)

(2) Interpretation of gathered context data

(3) Decision to recommend one service the migration to another node and modification of containment properties for a second service

(4) Physical movement of a service as consequence of the recommendation

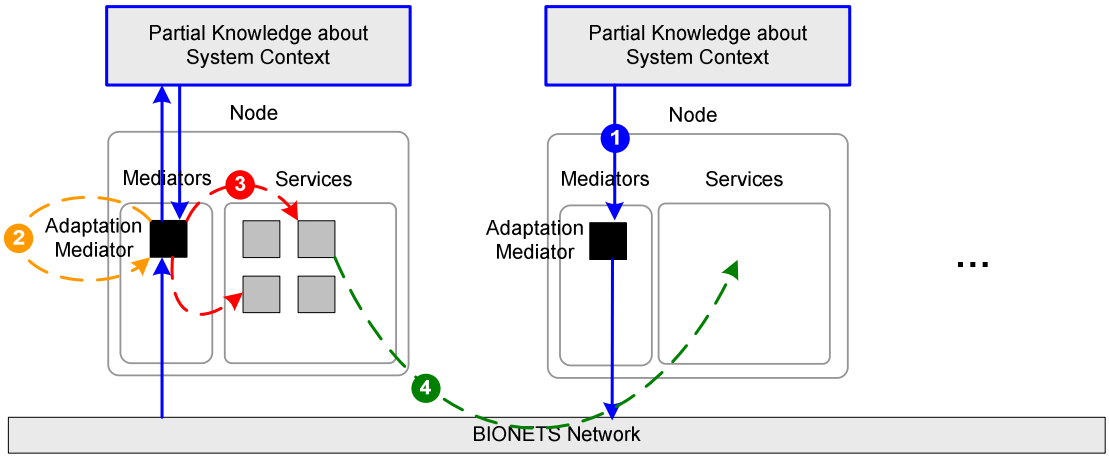

Figure 6: Example adaptation procedure

3.3.1 Self-aggregation of Service Cells and Service Individuals

Autonomic adaptation is understood as the possibility for a component to change its self-model and representation. This change is triggered by an entity that is able to identify the situations that determine the change. Aggregation algorithms can be seen as another way for the components to be autonomic. In this case, the change does not affect the self-model of an autonomic component, but the relationship with its neighbours: i.e. it addresses service composition. This can still be considered a form of evolution/adaptation to be handled by an entity subcomponent on the basis of the algorithms of self-aggregation.

In BIONETS system the Service Mediator can trigger aggregation for example for following reasons:

- The reasoner realizes that the load of the components to achieve its goal should be split among various others in order to improve the overall performance of the system.
- The component already belongs to an aggregation but the reasoner realizes that it cannot reach the component neighbours. In this case, the aggregation algorithms need to be started in order to recreate the aggregation itself.
- The reasoner realizes that, in order to achieve its current goal, the component needs a number of goals that it cannot achieve by itself. In this case, the

aggregation algorithm can be started to establish a stable relationship with those neighbours able to offer the required goals.

Various other reasons for aggregation can be identified depending on the specific application the components are built for. The reasoner will offer a language for expressing the rules triggering the execution of the aggregation algorithms. This language will be used by the developer to properly define the autonomic behaviour of components. The approach based on semantic and semantic reasoning is the one that is proposed.

An aggregated component (service individual or aggregation of service cell or of other service individuals) is one whose goal is obtained by composing the goals of single components: the result of an aggregation algorithm can be seen as a change in the self-model of the corresponding aggregated component. This allows to re-conduct all extensions to the Reasoner needed to support aggregation into the framework that considers any autonomic behaviour as a change in the self-model. The semantic and semantic representation of those aspects can cover the requirements for the language for triggering aggregation.

3.3.2 Requirements for a language supporting Self-aggregation

During discovery, it is essential to describe the capabilities of services, where such capabilities specify the overall functionalities of the service, and what the service does or provides. Expressing capabilities allows the whole interaction process to be goal-directed, so that the client can achieve the results it needs.

The representation of capabilities goes beyond the specification of the functionalities, and it should include the specification of the conditions under which the service works as expected, and what conditions it imposes on the requester. In addition to representing the different aspects of the capabilities of a service, a number of additional features, often called service parameters, need to be represented. These service parameters include things such as the cost model of the service, whether the service uses a pay-per-use model or a flat fee or whether the access to this service is restricted to a specific class of clients.

Since services have many different aspects, there is the need for a flexible representation schema that allows the service providers to express all the information that they need to express about their services, and the service requesters to express exactly what kind of services do they expect. To support such flexibility, any service representation schema should be based on an inference mechanism that allows the extraction of knowledge that is implicitly stated in the description, but not explicitly exposed, and the derivation should allow the service requesters to “reason” about the service specifications that they receive.

Such kinds of requirements will be further addressed in the forthcoming D.3.2.3 and D.3.2.4 documents.

3.4 Service evolution

The aim of this sub-section is just to emphasize that BIONETS Services are subject to evolution, and how this can be done. As illustrated in [BIONETSD322], the BIONETS system incorporated many challenges e.g. how it is possible to track the chances in the environment accurately enough in order to benefit from the evolution, and how rapidly it is possible to create a new service in evolution process. The [BIONETSD322] gives also a generic factors which limits the dynamics of evolution,

such as fitness estimation, mobility of nodes and services, and finally a speed of convergence of the specific evolution algorithm. Considering these limits, especially the two first mentioned are heavily related also to the overall service architecture framework. Thus, the framework should be designed in such a way that the effects of these limitations for service evolution can be minimized. In the following paragraphs some basic considerations of the services are given towards the proper solutions. Also the deliverable [BIONETSD322] gives on more detailed insight for the evolution and evaluation strategies.

A service can be a composite service individual, meaning that evolving such a composite service changes the way it is organised, i.e. its composition. The composition can be represented as a graph. A node in this graph represents a service, having as many entry points as different interfaces it offers, and as many output points as other service interfaces it must invoke in order to fulfill its job. The graph of a composite service makes it clear how the composite service delegates or distributes its job to some included services: edges between service nodes make explicit what are the service dependencies. Evolution of a service individual means replacing some nodes with some other and consequently modifying the dependency relations.

Each node can itself represent a composite service, so that the model supports recursive service composition. As we already mentioned, the service composition can be understood as service aggregation, and the process that drives such aggregation is useful both at initial service individual creation and for evolution of already existing service individuals.

The smallest blocks that constitute services are service cells. In this case, evolution means something different: it pertains to the transformation of the parameters or of the code that defines a service cell. The adaptation or evolution capabilities of service cells have been presented in Section 3.3.1, so will not be recalled here.

4. Network interface requirements

In order to cope more efficiently with the network communication, data and control messaging through the whole system, the control and data planes are divided as illustrated in the following Figure 7, the Service Communication Framework. In the communication framework, the data plane of the system provides the means for services to transfer the actual application/service data through the BIONETS network. The control plane *vice versa* provides the control and management functionalities over the whole system. In Figure 7, the control and data bearers (communication lines) are divided in the way that the red arrows represent the management and control bearers between different system blocks in the control plane and the blue arrows represent the data bearers in the data plane.

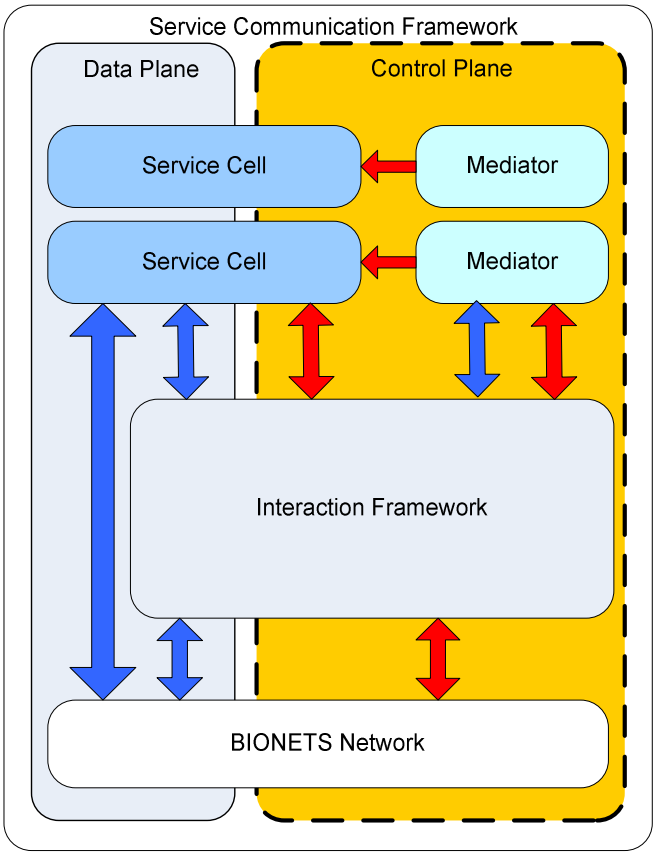

Figure 7: Service Communication Framework

Within the data plane, the Service Cells and Service Individuals can communicate either through the Interaction Framework (transferring semantic information through the Semantic Data Space) as described in the Section 2.1.3 in this document, or directly through the network. For example in the case of service migration the service needs to communicate directly through network for migrating to another node. Within the control plane the interaction between the “upper layer” service and network are carried out through the Interaction Framework as illustrated also in Figure 1. In order to provide control and management for the whole service life-cycle [D3.2.1], bidirectional control communication is required for the Mediators, Service Cells and Interaction Framework/Framework Footprint.

As mentioned in the earlier sections of this document, the mediator is responsible for service life-cycle management. Depending on the Service Cell and Individual concentration and type in each U-node, the functionalities of mediators need to be changed accordingly. Thus, this can be done purely through the evolution and adaptation of Mediator functionalities or by replacing and adding new Mediator components from the existing network. Due to the latter case, the mediator is also a part of the data plane and in this case the communication is performed via data bearers as in the case of Service Cells.

As presented in [BIONETSD122], the BIONETS system supports a set of basic communication primitives, which are used by different services. The communication primitives operate between the Interaction and Service Frameworks. The syntax for the basic communication messages for Interaction Framework and Mediator/Service Cells are defined in <attribute, value> –pairs, using the *SendMessage*(message) and *ReceiveMessage*(message) primitives. For the Interaction framework, the primitives *IfSend*(message) *and IfReceive*(message) are used. The attributes are not only used for identifying nodes, but also the data chunks and control messages in the system. In the [BIONETSD112], some examples for simple messaging and addressing are given.

For the Service Framework, the message types can be further divided into control and data messages, which are also used with (attribute, value) –pairs. The control and data messages are treated as datagrams. Although the communication in the BIONETS system is connectionless and asynchronous, in some cases the control messages need to be synchronized with operations and data delivery in order to provide near real-time communications, if necessary.

The control and data messages can be simply constructed from the (control, message) and (data, message) –pairs. For example in the case of triggering migration we can use message {<msgType, Control>, <TrgMigration, 1>}, in which the message type is set to control and the triggering of migration is set to 1, corresponding to launch the service migration either from Service Mediator or directly from the service's internal control loop [BIONETSD322]. For the data messages, a following structure can be used {<msgType, Data>,<dataType, temperature>} as illustrated in the [BIONETSD112].

The definition of the required messages are currently ongoing within the Service and Network Architecture Task Force. In order to cope with the disconnected nature of network infrastructure, one of the key control messages for the services and Service Mediator is the connectivity information of the network. For this a control message representing the connectivity of the each node inside the network island is required. In addition other side information can be also used, such as the location information of the nodes in order to arrange more efficient addressing of the data chunks in the BIONETS distributed system, and service and neighbor discovery.

5. Conclusions and future work

This deliverable presented the advances in the overall Service Architecture based on the work in WP3.2, WP1.2 and WP1.1. Several new ideas for Service Framework, Interaction Framework, semantic reasoning and Communication Framework were presented. The work with integrating the BIONETS Service and Network solutions to a common SerWork Architecture continues within the launched Service and Network Task Force. Also the impact of business models and security functionalities are currently under investigation and will be included in the next revision of the Service Architecture Description (BIONETS Deliverable D3.1.3).

The upcoming WP3.2 deliverables will also address more specific functionalities for semantic representation of services and data, and the different interaction framework models (namely semantic data space, publish/subscribe and distributed hash table solutions) their suitability and evaluation metrics. In addition the road map for service and node autonomicity needs to be clarified in more detail at the algorithm level for Service Mediators and Service Cells.

Terminology

Adaptation Mediator	A specialized Service Mediator handling the service adaptation routines.
AP-node	Access point to existing network infrastructure
BIONETS	BIOlogically-inspired autonomic NETworks and Services
CORBA	Common Object Request Broker Architecture
CRUD	Create, read, update and delete
Interaction Framework	Shared data space for carrying out the service cell and individual interaction as well as interface for network. Depending on the service requirements and communication paradigms the implementation of the Interaction Framework can vary from Semantic Data Space to Distributed Hash Table and Subscribe/Publish paradigms.
IOPE	Input, output, preconditions and effects
SDS	Semantic Data Space
Service	A BIONETS architecture entity offering services to users and other services.
Service Cell	Atomic service unit in BIONETS architecture
Service Individual	Blueprint for a composition of Service Cells
Service Mediator	Management entity for Service Cells and Individuals, provides the management and evolutions oriented functionalities.
SerWork	A higher level architecture containing both BIONETS network and service architectures
SOA	Service Oriented Architecture
T-node	Simple sensor nodes
U-node	Complex nodes offering services to users

References

[ACMN07] S. Alouf, I. Carreras, D. Miorandi, G. Neglia, Evolutionary Epidemic Routing, INRIA Research Report RR-6140, 2007-06-07

[LRSJ06] D. Linner, I. Radusch, S. Steglich, C. Jacob, Loosely Coupled Service Provisioning in Dynamic Computing Environments, in *Proc*. of ChinaCom, Oct. 2006

[JM06] B. Jennings, P. Malone, G. Gaughan, Charging for dynamically composed services in the Digital Business Ecosystem", eChallenges 2005.

[KCLOW05] J. Keeney, K. Carey, D. Lewis, D. O'Sullivan, V. Wade, Ontology-based Semantics for Composable Autonomic Elements, in *Proc*. of Workshop on AI in Autonomic Communications at 19th International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, 30/07 - 05/08 2005, 2005

[CASCADASD31] CASCADAS Project – Deliverable 3.1 - Aggregation Algorithms, Overlay Dynamics and Implications for Self-Organised Distributed Systems

[BIONETSD311] BIONETS Project – Deliverable D3.1.1 Service Architecture: Requirement Specification and concept definition

[BIONETSD321] BIONETS Project – Deliverable D3.2.1 Specification of Service Life-cycle

[BIONETSD322] BIONETS Project – Deliverable D3.2.2 Autonomic Service Life-Cycle and Service Ecosystems

[BIONETSD112] BIONETS Project – Deliverable D1.1.2 Architecture, Scenario and Requirements Refinements

[BIONETSD122] BIONETS Project – Deliverable D1.2.2 Disappearing Network, Autonomic Operation and Evolution.

[SBD07] J. Sacha, B. Bizkupski, D. Dahlem, R. Cunningham, J. Dowling, R. Meier, A Service-Oriented Peer-to-Peer Architecture for a Digital Ecosystem, in IEEE Int. Conf. on Digital Ecosystems and Technologies (DEST'07), 2007

[PVE05] G. Polyzos, C. Ververidis, E. Efstathiou, Service Discovery and Provision for Autonomic Mobile Computing, in 2nd IFIP Workshop on Autonomic Communication, LNCS 3854, 2005

[GBS03] P. Grace, G. Blair, S. Samuel, ReMMoC: A Reflective Middleware to Support Mobile Client Interoperability in Int. Symposium on Distributed Objects and Applications (DOA), 2003

[HLD04] C. Herault, S. Lecomte, T. Delot, New Technical Services Using the Component Model for Applications in Heterogeneous Environment, in LNCS 3473, Workshop of Innovative Internet Community Systems 2004, revised papers, 2006

[RM06] R. Rouvoy, P. Merle, Using Microcomponents and Design Patterns to Build Evolutionary Transaction Services, in Int. ERCIM Workshop on Software Evolution (EVOL'06), 2006