BIONETS

WP 1.1/3.1 – D1.1.3/3.1.3

SerWorks Architecture v. 1.0

Reference:	BIONETS/CN/WP1.1/3.1
Category:	Deliverable
Editor:	Daniele Miorandi (CN)
Author(s):	Jyrki Huusko (VTT), Francesco De Pellegrini (CN), Heiko Pfeffer (TUB), David Linner (TUB), Corrado Moiso (TI), Daniel Schreckling (HITeC/ UNIPASSAU), Daniele Miorandi (CN)
Verification:	Ludovic Henrio (INRIA), Iacopo Carreras (CN)
Date:	5/8/2008
Status:	Final
Availability:	Public

Executive Summary

In this deliverable, we introduce the first version of the SerWorks architecture, presenting the main SerWorks concepts and components, together with a description of the way the components interact. In SerWorks, a unified service-oriented architecture is applied to both application- and network-level functionalities. SerWorks supports the run-time generation of network protocols according to both the current environmental conditions and the requirements of the running services. Adaptation and evolution at the network level are driven by the running services, achieving extreme flexibility in terms of features the BIONETS disappearing network can support. This deliverable is the result of the merging of the activities of WP1.1 and WP3.1, and leverages the results achieved in Y2 by the SerWorks Task Force.

SerWorks enables a unified representation of network- and service-level BIONETS components, based on three main features: modularity, reusability and late binding. The main benefit foreseen for SerWorks-type architectures relates to the inherent flexibility added to the system in terms of network-level operations, which are not any longer statically designed and can be changed at run-time in response to either varying operating conditions as well as specific needs of the running services.

The SerWorks architecture is divided coarsely into three frameworks. The upper layer of the architecture is the Service Framework, which includes the application/service-level logic and the functions supporting their distributed and autonomic execution and management. The middle layer includes the Interaction Framework, whose purpose is to provide multiple concurrent interaction models supporting the communication among the distributed services and the realization of a shared data space. The lowest layer is the Networking Framework, which provides the basic communication capabilities in the disappearing network context, supporting means for establishing secure communications over opportunistic networks.

The Service Framework provides a runtime environment for the execution of service logic running on U-nodes and T-nodes. On the other hand, the service framework includes capabilities to react in an autonomic way to changes in the context/environment. These capabilities include different evolution and adaptation strategies, service life-cycle handling and mobility support. The necessary control logic for performing such functions is abstracted into service mediators, architectural elements complementary to the service cells.

Services and service mediators interact through the Interaction Framework, which supports a variety of interaction models, including, e.g., the Semantic Data Space. The Interaction Framework decouples the Service Framework from the underlying communication protocols, naming/addressing schemata and network characteristics.

The networking framework aims at provide appropriate communication means for effectively supporting networked services in the presence of large-scale, heterogeneous and often partitioned networks. Traditionally, network architectures have been built according to the layered OSI model, based on a stack structure. Each layer of the stack can be implemented in a separated fashion and independently of the other ones. The BIONETS networking framework is built around a non-layered and dynamic architecture, Functionalities are encapsulated in components that can be dynamically bound and composed. Interoperability is ensured by separation of the functional and non-functional aspects of network-level components.

This document is meant to set the baseline for the algorithms and methods to be developed in WP1.2 and WP3.2. The second iteration of the SerWorks architecture, expected at M42, will profit from the understanding gained through prototyping and experimental activities carried out within WP5.

Document History

Version History

Version	***Status***	***Date***	***Author(s)***
0.1	*Created*	*15/5/08*	*Daniele Miorandi*
0.2	*Draft*	*12/6/08*	*First completed draft*
0.3	*Draft*	*25/6/08*	*Daniele Miorandi*
0.4	*Draft*	*3/7/08*	*Daniele Miorandi*
0.5	*Draft*	*24/7/08*	*Heiko Pfeffer*
0.6	*Draft*	*28/7/08*	*Daniele Miorandi*
1.0	*Final*	*5/8/08*	*Daniele Miorandi*

Summary of Changes

Version	***Section(s)***	***Synopsis of Change***
0.1	*Not Applicable*	*Template and initial Table of Contents drawn*
0.2	*All*	*Added contributions from all partners*
0.3	*All*	*Revised version of contributions and alignment*
0.4	*All*	*Harmonization*
0.5	*2.2,2.3*	*Revision*
0.6	*Summary, 1,2*	*Revision*
1.0	*All*	*Harmonization*

Contents

1. Motivation and Relevance

The BIONETS project stems from the observation that nature presents a long record of successes in dealing with heterogeneity, scalability, complexity and dynamism issues in systems' organization. There are plenty of examples of large-scale ecosystems which are able to self-organize and co-evolve in such a way to reach efficient equilibria while being able to adapt to varying environmental conditions. By replicating, in a digital system, the underlying biological processes, the project aims at devising innovative strategies for engineering autonomic computing and communication systems.

In BIONETS, heterogeneity and scalability are mainly tackled at the network architectural level. Indeed, we rely on an architecture which is inherently scalable and at the same time is designed in such a way to natively accommodate devices heterogeneity [BIONETS_D111, BIONETS_D112]. Complexity and dynamism issues are addressed through the introduction of self-evolving autonomic services, built around concepts and tools inspired by the functioning of biological systems. At the same time, one of the most innovative aspects of the BIONETS project is given by the notion of merging networks and services in what we call *SerWorks*. In SerWorks, a unified service-oriented architecture is applied to both application- and network-level functionalities. SerWorks aims at enhancing the flexibility provided by current network-level solutions by enabling adaptation and evolution of network protocols to be driven by the running services. In SerWorks, service-tailored network protocols can be built on-the-fly according to service constraints and environmental features.

In this work, we present the main concepts and components of the SerWorks architecture, together with a description of the way the components interact in order to allow system's operations.

The BIONETS system architecture is built around two basic classes of devices [Carreras07]. The T-nodes are meant to perform simple reactive tasks (like, e.g., sensing) and could represent sensors, tags, etc. The U-nodes represent an abstraction of personal user devices, e.g., notebooks or mobile phones. T-nodes and U-nodes communicate wirelessly, forming transient islands of connectivity. In this regard, the BIONETS system architecture was designed to address the following requirements:

- Enabling, in a disappearing and mobile network context, the execution of distributed cooperative services, involving service cells and data deployed/available on different nodes;
- Supporting dynamic evolution and adaptation of services in order to address changes in the execution context (e.g., network disconnections, changes in resource availability, lack of data/information);
- Providing means for dynamic aggregation, collaboration and composition of services and data.

The SerWorks architecture is divided coarsely into three frameworks. The upper layer of the architecture is the Service Framework, which includes the application/service-level logic and the functions supporting their distributed and autonomic execution and management. The middle layer includes the Interaction Framework, whose purpose is to provide multiple concurrent interaction models supporting the communication among the distributed services and the realization of a shared data space. The lowest layer is the Networking Framework, which provides the basic communication capabilities in the disappearing network context, supporting means for establishing secure communications over opportunistic networks.

In BIONETS, services are defined as entities that may provide knowledge, content, or functionality to other services and users. User applications, application services, network-level services and protocols can therefore be treated through such a single architectural concept. Services are described as self-similar compositions of other services and service cells, which are atomic logic elements. The Service Framework provides a runtime environment for the execution of service logic running on U-nodes and T-nodes. On the other hand, the service framework includes capabilities to react in an autonomic way to changes in the environment of services. These capabilities include different evolution and adaptation strategies, service life-cycle handling and mobility support [BIONETS_D321].

Autonomicity in BIONETS is broadly understood as the ability of services to present the self-CHOP (-configuration, -healing, -optimization, -protection) features identified as fundamental traits in the IBM Autonomic Computing manifesto [Kephart03]. In BIONETS, autonomicity is provided on the node level [BIONETS_D312]. Specific components (called mediators) are deployed on U-Nodes and address the aspects of autonomic life cycle management on behalf of the services.

Services and service mediators interact through the Interaction Framework, which supports a variety of interaction models, including, e.g., Semantic Data Space (SDS) [BIONETS_D321] and publish/subscribe. The Interaction Framework decouples the Service Framework from the underlying communication protocols, naming/addressing schemata and network characteristics, e.g., the disappearance of nodes from a connected island.

Network interfaces provide communication primitives, implemented in order to cope with the disappearing network context. The main objective of the networking framework, as detailed in the following sections, is to provide appropriate means for effectively supporting networked services in a disappearing network environment. Traditionally, network architectures have been built according to the layered OSI model, based on a stack structure. Each layer of the stack can be implemented in a separated fashion and independently of the other ones. Typically, applications leverage the communication pipe trough a set of executable APIs, whereas the communication/networking functionalities are part of the operating system. The layered approach has a clear advantage in that it enables (at least in principle) progressive deployment of new solutions. On the other hand, the limit is that, in current systems, the stack is hardwired in the OS and is not flexible.

We would like to stress the two main features that traditionally differentiate communication protocols compared, for example, to threads occurring among processes of a same computation or high-level services composed by instances of different programs or modules possibly running on different devices. First, communication protocols have a top-down sequential architecture where the thread occurring at the upper layer (e.g., the application layer) has to be supported by the underlying threads down to the physical bit transmission [vanRenesse95]: packets are passed down to the layer in charge of the “next” step until electromagnetic propagation occurs at the bit level. Thus, the overall flow is loop-free and sequential. Second, current architectures usually use at most one block at each layer at the time: for example, once established, a socket leveraging UDP cannot switch to TCP since the peer UDP module would fail in treating packets of the new session according to TCP.

Nevertheless, non-layered and dynamic architectures are possible: the advantages of dynamic and/or amorphous network protocol architectures have been addressed by the research community [Tschduin91, OMalley92, Braden02]. At the same time, such solutions have not made their way into widely deployed network software and the OSI-like stack paradigm is *the* way communication protocols are conceived and implemented. There seems to be two main factors limiting the use of dynamic architectures, in which basic building blocks get composed at run-time. The first one is computing power/efficiency: communication protocols require to carry out operations at the speed at which transmissions on the physical links are possible, which can easily be of the order of hundred Mb/s. The second one is the lack of a suitable distributed run-time execution environment able to support such dynamic reconfigurable architecture. It is our belief that such blocking factors will become less prominent in the future, and that such solutions will become attractive when applied to highly dynamic mobile environments, such as the BIONETS disappearing network settings.

Interoperability issues are addressed by defining components with well-specified functionalities and interfaces (equivalent to “roles” in role-based architectures [Braden02]) and by separating functional and non-functional aspects. Dynamic binding is used for changing at run-time the non-functional aspects of components, while the functional ones remain unchanged. As an example, a component in charge of data dissemination will change the algorithm used for relaying/forwarding packets, but not the way it can be triggered upon reception of a packet and/or presence of a new neighboring node.

The *SerWorks* vision builds on a service-oriented approach to networking protocols. Protocols can indeed be understood as the composition of various networking services. This makes it possible to build a flexible architecture, in that network-level services could require input from several other services, and provide different outputs to several services. In a first approach, we may retain the functional separation between the service and the network framework. In the latter one, network services may get dynamically composed at run-time, according to the (functional and non-functional) requirements as determined by the service mediator. Such information will be used by a networking mediator to decide on the best composition of the available network services. From the service point of view, the advantage is clear: the possibility of exploiting a flexible data bearer. The latter one can

be though as an “ad hoc” network protocol stack, offering socket-like functionality to the interaction framework but tailored to the specific service needs.
The resulting architecture would encompass mediators at both the service and the network level. In order to optimize the resulting architecture, we envision, in a future extension, to join the two entities. In this fully integrated SerWorks architecture, three different “layers” are present. They expose well-defined APIs to the upper/lower one; on the other hand, the definition of the modules performing the actual functions is left to the mediators which can decide at run-time the optimal joint configuration for the service, interaction and networking frameworks.
This document is organized as follows. In Sec. 2, we detail the SerWorks architecture and its functional components. Sec. 3 describes the possible interactions among the SerWorks components, including a characterization of interfaces among the various frameworks involved and a description of typical interaction flows. Sec. 5 concludes the deliverable. App. A includes a list and description of the terms used throughout the deliverable. App. B presents the interfaces used for service discovery and adaptation. App. C reports an example, based on the “Unsolvable Quiz” demo developed by the project consortium.

2. Architecture and Components

2.1 High-Level Architecture

In this section we will present the high-level architecture of BIONETS SerWorks, which combines the network and service frameworks to a single architecture. The architecture is defined by the specific macro-functionalities on both service and network side. These macro-blocks are shortly defined and discussed in this section and the more detailed functional descriptions are given in the following sections.

2.1.1 Basic Service Framework Concepts

As described already in [BIONETS_D311, BIONETS_D312], the main building blocks of the service framework are Service Cells (SC)/Service Individuals (SI) and Service Mediators. In the BIONETS framework, one Service Mediator is bond to certain Service Cells sharing the same management functionalities. The interaction framework provides a means for Service Mediators to communicate with each other within the network island. The purpose of the Interaction Framework is to provide flexibility in terms of support of multiple concurrent and evolvable interaction models, including, e.g., publish/subscribe, blackboard etc. The current BIONETS SerWork architecture is build to support Semantic Data Space [BIONETS_D321] as default interaction model. The service framework can be depicted according to these building blocks as in Figure 1.
The T-nodes do not, in general, support a full-fledged Interaction Framework. T-nodes implement in any case a minimal Interaction Framework footprint in order to, e.g., advertise and access the sensor data and T-node services.

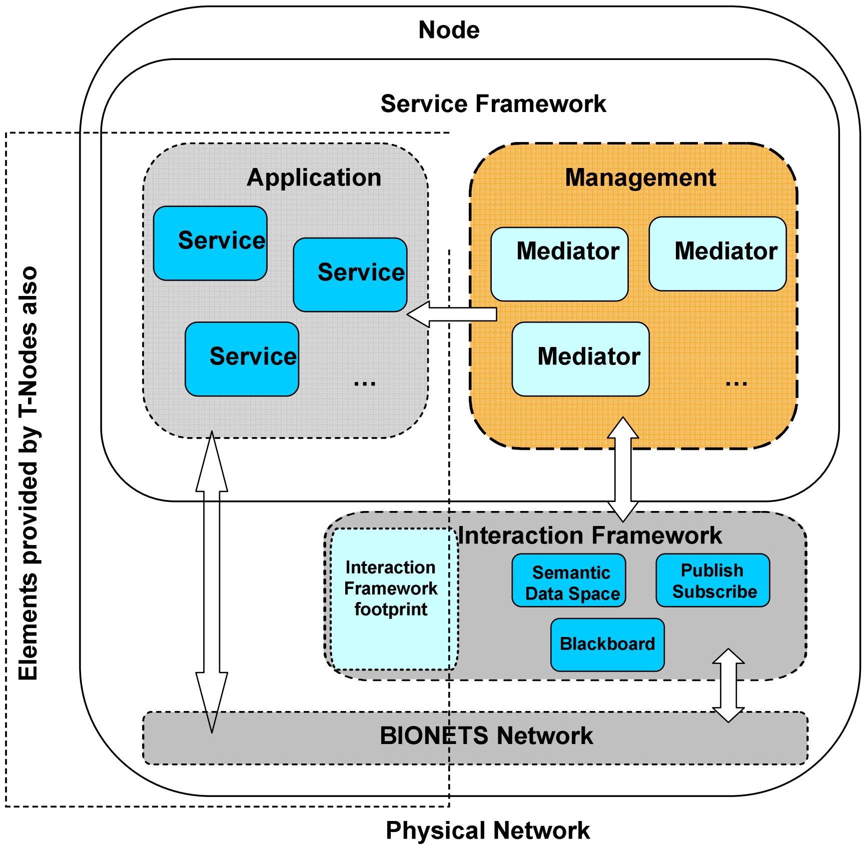


Figure 1. SerWork Service Framework

Communications in BIONETS SerWorks are represented in Figure 2. The communication functionalities are divided into data and control plane operations. Similarly, the communication lines are divided into data and control bearers. The control plane includes control and management protocols and functionalities, and handles mainly security and communications among mediators and service cells. The data plane and data bearers correspond to the data delivery protocols and functionalities and handle the application/service data communication between nodes through the network framework. Figure 1 and Figure 2 illustrate mainly the data-oriented part of the system, leaving the user and application service requests out of the scope of this deliverable. However, as described in the previous deliverables on service architecture [BIONETS_D311, BIONETS_D3.1.2], the BIONETS service framework includes interfaces for service requests through standard service-oriented architecture.

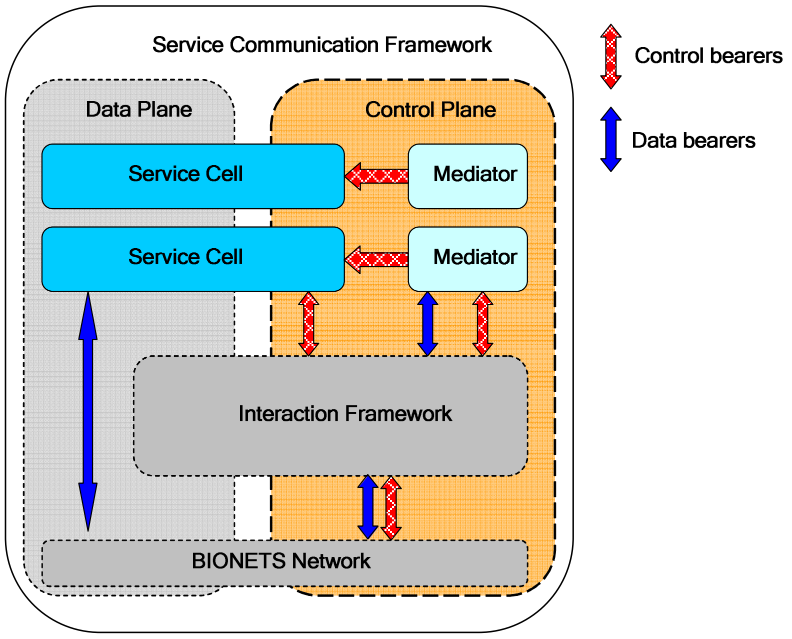


Figure 2. SerWorks Service communication framework.

2.1.2 Basic Network Concepts

The current definition of the BIONETS network framework includes eight different functional components as illustrated in Fig. 3. The main components are:

1. Data dissemination
2. Interoperability with IP
3. Naming system
4. Information filtering
5. Information gathering
6. Opportunistic communications
7. Secure Communications
8. Peer-to-Peer Cloud

These functional components are described in more detail in Sec. 2.4.
Each node or service is allowed to dynamically switch between different breeds of components but the functional components themselves are mandatory for BIONETS node to work. In the same way these components cannot be distributed over different nodes.

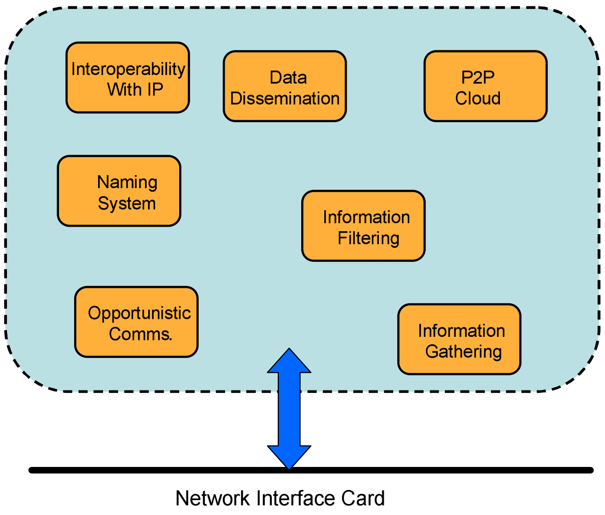


Figure 3. Network framework functional components

One of the most important parts of the network components in terms of Service and Network architecture integration is the P2P Cloud component, which creates a virtual overlay through which the service can access the network resources. The overlay provides also the connectivity support within the BIONETS cloud needed to perform efficiently migration of service cells (see App. B for details).

2.1.3 SerWorks – Incremental approach

The previous sections illustrated the main components of the BIONETS service and network frameworks. This section defines the development strategy to be followed for SerWorks development. In the current SerWorks architecture the service framework supports well modularity and dynamicity. Similarly in the network framework, the modularity is also supported (components), but the system lacks slightly of dynamicity and cognitive behaviour. In order to enhance the whole system a similar approach can be used for network as for the service framework. In the first step (step-1) specific mediators for management and control of the network components will be introduced, as illustrated in Figure 4. The main idea is to utilize specific network component mediators in the network control plane in order to supervise and control the component and protocol behaviour.
In the second step, unique Mediators and common control plane for both network and services will be introduced. This is illustrated in Figure 5. Transition to step-2 requires additional work for mediator implementation due to the increased complexity on control and management plane and mediator interactions - especially for small devices - as also noted in [Frantti08] for distributed control and management planes Services themselves (application and network components) should stay unmodified. The arrows in Figure 4 and Figure 5 represent the interactions between the main components. In the network framework, the mediators are used for controlling the network components. The direct interactions between mediators and network components are expected to be mostly unidirectional; bidirectional interactions are handled through the interaction framework and network framework. In Figure 4, the red bidirectional arrow with pattern represents the interaction between the network components and interaction framework/service framework data plane. The blue bidirectional arrow represents the control and management plane interactions between the network framework and the interaction framework.
Finally, the second step of this approach is also very important for the security framework embedded into SerWorks. Special Mediators can control and construct services which provide certain security functionalities using different Service Cells (SC) from different layers. E.g. a service may require secure communication over the network as it distributes its service over several nodes. A Security Mediator may now use network security components on the networking layer. Additionally, some parts of the same service may require application layer security. The mediator is also able to access security components on this layer and can thus control an efficient service execution.

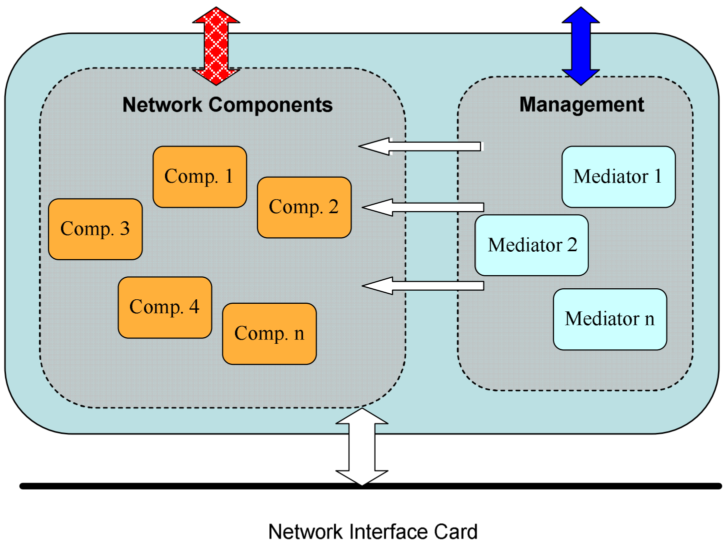


Figure 4. Step-1: Network component Mediators

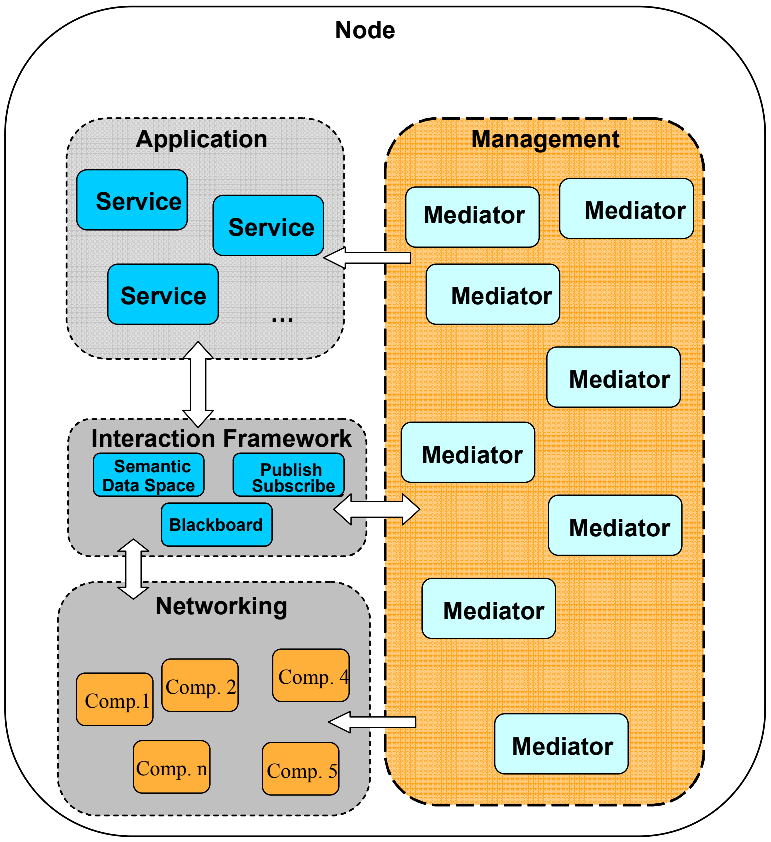


Figure 5. SerWorks development step-2: common management plane

2.2 Architectural Components

This section outlines the key components of the BIONETS architecture. Therefore, a refinement of the BIONETS service life-cycle introduced in deliverable D3.2.1 [BIONETS_D321] is outlined in

Sec. 2.2.1, while the mediators required to support the related functionality is derived and presented in Sec. 2.2.2.

2.2.1 Refinement of the BIONETS Service Life-Cycle

The refinement of the BIONETS service life-cycle is depicted in Figure 6; here, each life-cycle run is always coupled to a specific request. Thus, as soon as a request is created by the user, the life-cycle is within one of the depicted *states*, performing a specific *action*. Based on certain *events*, the service moves into another state of its life-cycle; those events are marked by reference points with numeric labels (A-H).

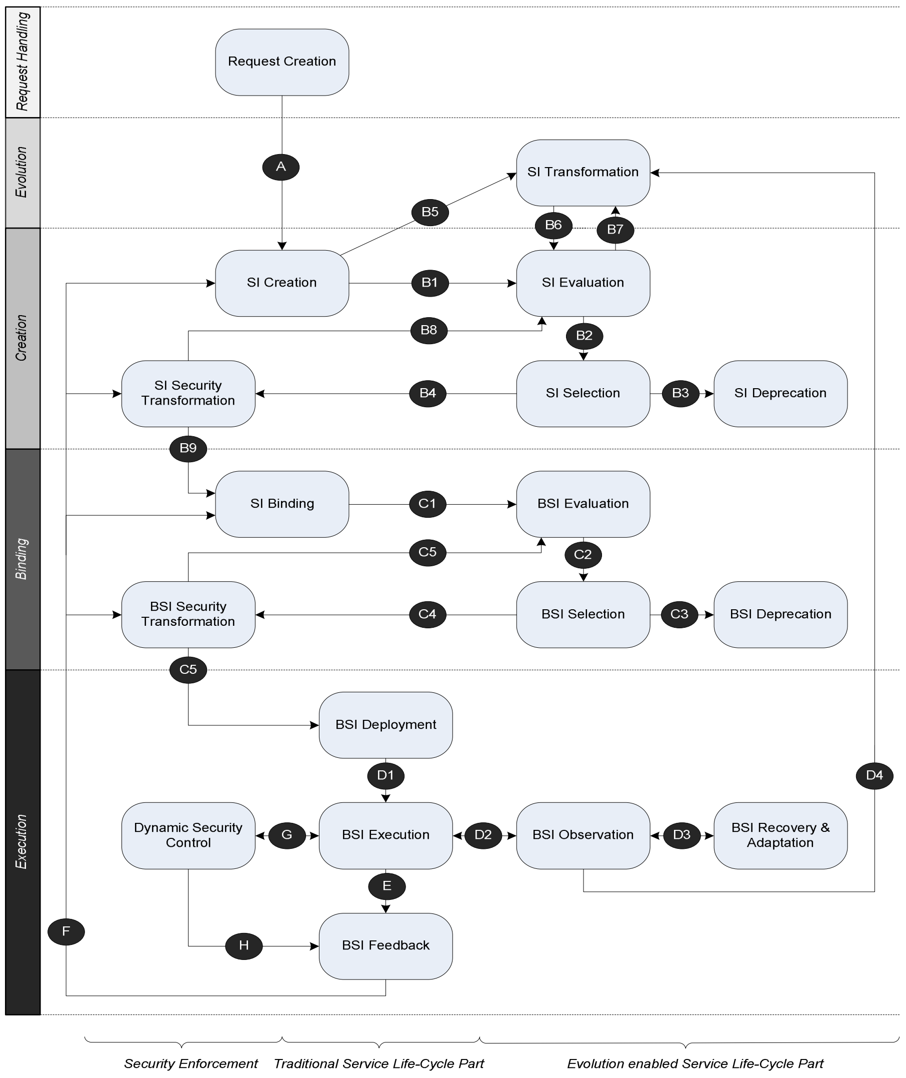


Figure 6: BIONETS Refined Service Life-Cycle

Within the following, we discuss the single steps of the BIONETS service life-cycle.
Initially, a request is created by the user and passed to the Service Individual (SI) Creation state (A). Here, the request of the user is transformed into a machine readable format, thus, a Service Individual. This creation can be achieved in three ways.

(1) An appropriate SI matching the given request is already available.

(2) An appropriate SI is dynamically created by an algorithm for dynamic service composition.

(3) An appropriate SI can possibly be derived by the modification of already existing SIs.

For case (1) and (2), all SIs (either directly available or derived trough a service composition approach) possibly matching the given request are passed to the SI Evaluation state (B1) where their fitness with regard to both the functional and non-functional aspects is evaluated. Here, the *fitness* of a SI refers to a unique value that can be clearly compared to all other possible fitness values. The simplest way to represent a fitness of a service would be a number between zero and one, where zero stands for a very unfit SI while a fitness of one represents the best SI possible. Moreover, every other representation of a SI's fitness is possible, as long as the question "$x>y$?" remains decidable for every two fitness values x and y. The derivation of the fitness value is mainly twofold. On the one hand, the *functional fitness evaluation* estimates how good the functionality provided by the SI fits the functionality requested by the user. The *non-functional fitness evaluation* on the other hand gives evidence of a SI's quality with regard to its non-functional properties such as resource consumption, trustworthiness or rating within a community of users. Research on fitness evaluation is currently ongoing; results will be presented in deliverable D3.2.5 (due at M36).

Within the subsequent SI Selection state (B2), the fittest SIs are selected (B4) while the remaining ones are deprecated (B3). The number of SIs to be selected can be defined by simple rules. For instance - in case the fitness is represented by a value between zero and one - all SIs with a fitness above 0.8 can be selected. Alternatively, the 3 best SI may be chosen. In general, it was observed that a limitation in the selected SIs eased the speed of the service transformation and evaluation procedures since only a limited number of SIs is processed in parallel. Simulation settings for this issue are introduced in deliverable D3.2.4 while related results can be found in deliverable D3.4.1. For case (3), existing SIs are selected for a transformation procedure (B5) that aims at modifying existing SIs by the application of genetic operators in order to evolve their functionality towards a better fit for the given request. First results on this evolution of SIs through genetic operators have been published in [Linner07]. The SIs resulting from this evolutionary state are then evaluated (B6); based on their fitness, either a second evolution iteration can be applied (B7) or the previously described selection procedure continues, some SIs being selected and the other ones deprecated. If no appropriate SI can be created within several evolutionary iterations, all SIs may be deprecated; the user request is then answered to be unsolvable.

The selected SIs are passed on to the SI Security Transformation state (B4). In order to ensure that the service individual complies with the local security policies of the service, node, user and data possibly involved in the execution of the service, feasible analysis methods are applied in order to adjust the SI to the appropriate security requirements. After this analysis and the corresponding transformation steps, the SI needs to be evaluated again in state (B8). The security transformation is only based on security policies. As the SI still has to fulfill the user request the transformations are designed to generally not change its functional and non-functional aspects. By trying to separate these aspects, security transformations will generally not interfere with the original functionality of the SI. In some cases this may not be avoidable though, e.g. if a security mechanism requires direct user interaction. Finally, some transformations may render the SI unusable, e.g., a user may not possess the appropriate credentials to access a certain data item on the node. An execution of the service in this situation would simply fail. Additionally, the SI has to respect the current context, e.g. resources availability. Thus, some service may request security mechanisms that the current context cannot support. These SI may be deprecated too in state (B3). The remaining SIs become possible candidates for execution.

They are passed on to the SI Binding state (B9). Here, all Service Composition Blueprints (SCBs) of the selected SIs should be bound to SCs, i.e. real service implementations. Therefore, the service descriptions of the SCBs are matched to those of the both local and remotely available SCs. Assuming that n SIs have been passed to the SI Binding state, the result of this binding procedure may constitute $n \times (m_1, m_2, \ldots, m_n)$ Bound Service Individuals (BSIs) (given that m_i denotes the number of different possible bindings for the i-th SI). The following states reached by (C1), (C2) and (C3), respectively, correspond to the ones for the treatment of the unbound SIs, i.e. the fitness of the BSIs is evaluated and the fittest ones are selected while the rest is deprecated. In contrast to the SI Creation phase, BSIs cannot be transformed by genetic operators, since they are defined for modifying the underlying structure of an individual, referred to as genotype within evolutionary biology. BSIs on the other hand are already instantiated individuals, so-called phenotypes, which can no longer be modified by genetic operators but can only adapt to their environmental context, limited by their binding to concrete service implementations. Although the binding procedure already respects certain security

characteristics of SIs, e.g. the exclusively local binding of SCs if the SI describes a security protocol, the selected BSIs might yield additional security issues due to the bound SCs. In state (C4) possible security solutions are iterated trying to adapt the BSI to comply with the security requirements. The transformed service is passed to evaluation in state (C5) comparable to the SI in state (B8). If the security check succeeds, the BSIs are passed for execution.
In case more than one BSI has been selected for deployment (D1), one BSI that should be finally executed is chosen either by random or through a user feedback. During service execution, an observation state is entered in case a service is no longer available or a change within the environmental context has been detected (D2). The BSI Observation state first tries to treat the according event by passing it to the BSI Recovery & Adaptation state (D3). Here, the configuration of a SC can be adapted or single SCBs can be recovered by a fast binding procedure in order to provide a continuation of the current service execution. In case recovery and adaptation mechanisms fail, genetic operators can again be applied to transform the respective SI (D4). If the environmental context changes or modifications in the Recovery & Adaptation state imply changes for the current security context, the BSI execution is still controlled by runtime security checks. After finishing service execution, a feedback of the BSI is calculated (E) and passed to the SI Creation and Binding Phase where the accumulated knowledge can be used for decision making procedures during the treatment of upcoming requests (F). Provided, environmental changes caused BSI changes or there are BSI execution paths which do not comply with local security policies, security exceptions are triggered in state (G). These exceptions can also provide important feedback (H) for later use in the Service Life Cycle.

2.2.2 Architectural Component Description

Within the following section, we derive the key Mediator components required to support the BIONETS service life-cycle as outlined in Sec. 2.2.1. An overview of the relation of the single core Mediators is given in Sec. 2.1 Note that the communication paths indicated between the single Mediators abstracts from the implementation. A possible realization based on the SDS introduced in deliverable D3.2.1 [BIONETS_D321] is again sketched in Sec. 2.3.

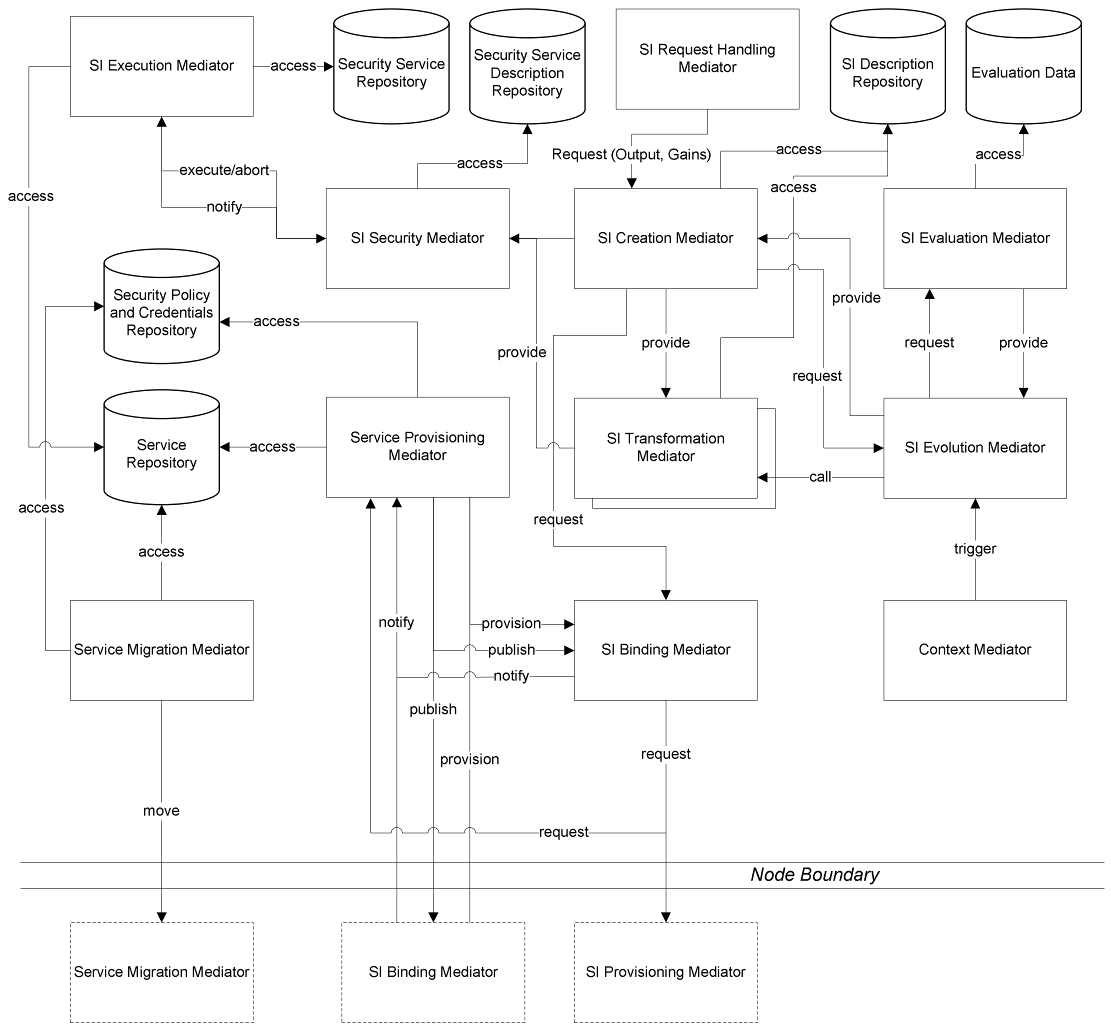


Figure 7: BIONETS Mediators: Overview

SI Request Handling Mediator – The *SI Request Handling Mediator* builds the interface to the user. Here, the request specified by the user is mapped to a final set of gains and outputs that are requested (see deliverable D3.2.4 [BIONETS_D324], specification of the service model). In the simplest realization, a GUI has to be presented that enables the selection of desired outputs and gains.
SI Creation Mediator – The *SI Creation Mediator* builds the core for the collaborative processing of user requests. It has access to the *SI Description Repository*, where appropriate SIs matching the request can be looked up. In case a suitable SI is found, the *SI Creation Mediator* requests bindings for each service blueprint from the *SI Binding Mediator* and finally passes the resulting BSI to the *SI Execution Mediator*. If no matching SI can be found, a mechanism for dynamic service composition is started, trying to automatically build a SI from available service descriptions. (The development of such an algorithm for dynamic service composition is under research and will be presented in deliverable D3.2.6 at month M42.)
Service Provisioning Mediator – Making available services for remote nodes is the key feature of the *Service Provisioning Mediator*. If the node and/or user policy comply, it publishes service descriptions at remote nodes (where they are stored in the *SI Description Repository*) and waits for an according request, given by a notification message. It then provisions the service to the given requester.
SI Binding Mediator – The *Service Binding Mediator* builds the counterpart to the *Service Provisioning Mediator*. It requests service blueprints from either the local or remote *Service Provisioning Mediators* by sending service requests messages. Thus, the mediator looks up the location of required services in the *SI Description Repository* and tries to reach the *Service*

Provisioning Mediator of the node hosting the service. (In case multiple nodes host appropriate services, the mediator tries to request the service from all nodes simultaneously.) The addressed *Service Provisioning Mediators* are informed by the service request by a notification message and confirm the service provisioning by an acknowledgement. The first node sending an ACK is selected for service provisioning.

SI Security Mediator – Before a bound service individual can be executed the *SI Security Mediator* applies a static check of the SI in order to verify whether the service complies with the local security policies. In case the assembly does not comply with the security requirements, this mediator tries to enhance the service individual with feasible security service individuals if possible or denies its execution. During execution the SI Security Mediator can abort the execution of the BSI if it performs unauthorized actions.

SI Execution Mediator – The *SI Execution Mediator* executes a BSI based on its workflow graph.

Context Mediator – The *Context Mediator* is the main driver for the evolution of services. It receives input from sensors and/or users about the environmental changes and sends well-defined messages to the *SI Evolution Mediator* in order to trigger evolutionary procedures.

SI Evolution Mediator – The *SI Evolution Mediator* is the centerpiece of service evolution. Here, evolutionary procedures are initiated based on one of the following triggers:

1. Message from the *Context Mediator*
2. Request from the *SI Creation Mediator* to evolve an already existing SI
3. Feedback from the *SI Evaluation Mediator*

Thus, based on incoming triggers, the *SI Evolution Mediator* can initiate the transformation of SIs. An interface to the *SI Evaluation Mediator* enables the estimation of a SIs fitness.

SI Evaluation Mediator – The *SI Evaluation Mediator* evaluates the fitness of a BSC with regard to its non-functional properties. An appropriate model for representing such properties in a lightweight way will be introduced in deliverable D3.2.5.

SI Transformation Mediator – The *SI Transformation Mediator* has to be considered as a placeholder for multiple Mediators capable of processing an evolutionary step on service level. One example is a Mediator applying genetic operators on service compositions as introduced in [BIONETS_D324].

Service Migration Mediator – The *Service Migration Mediator* enables the movement of service application logic between different nodes. An extensive description of the related migration procedures can be found in deliverable D3.2.4 [BIONETS_D324]. The migration is inhibited if it conflicts with the local security policies, e.g., if it conflicts with the users privacy policy.

Service Repository – Holds references/pointers to all locally available services.

Security Service Repository – Holds references/pointers to all locally available security services, e.g., authentication and key exchange protocols etc., as well as security service cells, e.g., encryption function, random number generators, hash functions etc.

Service Description Repository – Holds SIs as defined in deliverable D3.2.4 [BIONETS_D324]. Thus, a SI is given by its workflow graph, its dataflow graph and a finite set of service blueprints. (In case only one service blueprint is involved, the SI is a SC.) Note that the services specified by the service blueprints do not necessarily have to be hosted on the local node.

Security Service Description Repository – Holds Security SIs as defined in deliverable D3.2.4 [BIONETS_D324]. It is also given by its workflow graph, its dataflow graph and a finite set of service blueprints. (In case only one service blueprint is involved, the SI is a SC.) Note that the services specified by the service blueprints have to be hosted on the local node.

Security Policy & Credential Repository – Holds the security policies of the node and the user as well as their credentials.

2.3 Interaction Framework

The Interaction Framework was introduced in Deliverable D3.2.1. It serves as middle layer between networking and application services. The purpose of the interaction framework is to guide the design of special communication and coordination services that allow application services and service mediators smooth and open interworking. These communication and coordination services, also called Interaction Models, are built on a few, atomic operations for resource manipulation. In the original

description of the Interaction Framework these operations were Create, Read, Update and Delete (also abbreviated as CRUD). CRUD operations are supposed to be supported by the networking framework. In practical tests the Hypertext Transfer Protocol (HTTP) pointed out as appropriate example for a network-side CRUD realization.
In practice, the Interaction Framework on each node reflects a finite number of named resources that can be accessed by CRUD principles, but only the Interaction Framework on U-Nodes allows operations on resources of other nodes (T-Nodes are not allowed to initiate interactions themselves). The behaviour of these resources, i.e. their reaction to CRUD operations, is defined through respective Interaction Models. Thus, the Interaction Model introduces a meaning for the CRUD operations. Let's assume an example setup with two U-Nodes X and Y controlling a finite number of resources. In this case a publish/subscribe model would work in the following way:

1. U-Node X prepares a subscription by creating a local resource as placeholder for a later notification.
2. U-Node X creates at U-Node Y a resource that contains information about the kind of notification U-Node X is interested in and the name of the resource U-Node X created as local placeholder for feedback.
3. New information becomes available at U-Node Y; U-Node Y checks the type of information U-Node X was interested in and updates the resource that served as placeholder at U-Node X with the new information.

The BIONETS service architecture introduced one new Interaction Model, called Semantic Data Space (SDS). As explained above, the SDS is realized with the capabilities provided by the Interaction Framework, namely resource management through CRUD and the capability to do any computations on these resources. The SDS is an abstraction of the capabilities provided by the Interaction Framework and adds value while providing means for the coordination of joint tasks among Service Mediators. The SDS is a successor of the plain old Tuple-space model, but basically works in a similar manner. Multiple parallel processes (in BIONETS Service Mediators) use the SDS like a shared memory system to choreograph and synchronize their actions. Differently from the classical Tuple-space model, the data written to the SDS are no self-contained tuples, but rather nodes and edges of a data graph following a proper grammar. The SDS allows Service Mediators to open spaces, write, read and subscribe for data. Write operations that violate the grammar of the graph are rejected by the SDS. In such a way, various coordination concepts like, e.g., mutual exclusion or time division multiple access can be enforced. Furthermore, the graph model can also be used for access control. It can partition data in such a way that users, nodes, or groups which are not authorized to read or write specific data can be excluded. Additionally, each data item is equipped with a data security policy, as described in D4.2. The data security will allow read and write operations on data only if they comply with the data security policy. Finally, the SDS' capability to prove consistency of data against the grammar makes the SDS powerful. In D3.2.1 Resource Description Framework (RDF) and the Semantic Web language stack were proposed as grammars for data on the SDS. The SDS uses CRUD operations to mirror and synchronize its graph representation internally, across multiple nodes as depicted in Fig. 8.

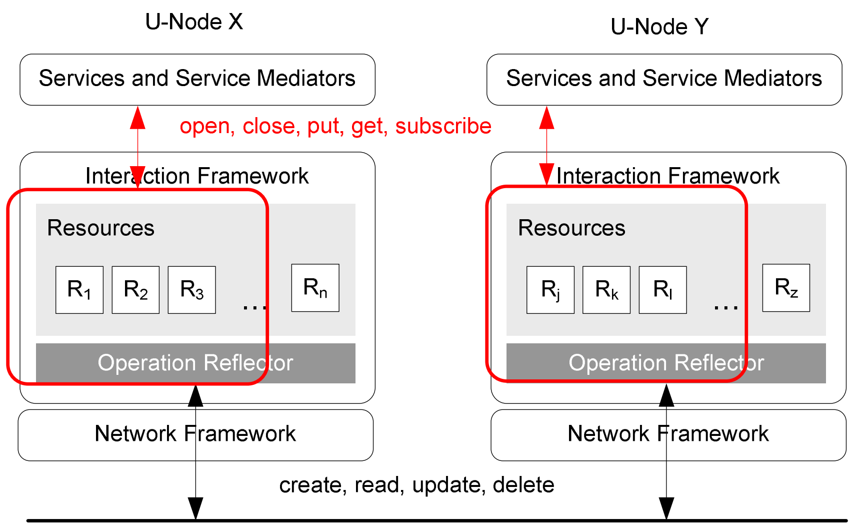


Figure 8: Role of CRUD and SDS in Interaction Framework

To Service Mediators the SDS provides the following seven operations:

Operation	**Description**
Data-Space-Handle := **open** (*Max-Idle-Time*)	*The SDS can contain multiple spaces, while each space contains data about a particular subject of interaction. The* ***open*** *operation opens a new data space and retrieves a reference to this space for further operations to the Service Mediator that requested the opening. A maximum idle time ensures that no obsolete data remain in SDS after the end or the cancellation of an interaction.*
close (*Data-Space-Handle*)	***close*** *simply closes a data space in the SDS.*
put (*RDF*, *Data-Space-Handle*)	***put*** *adds data to a data space. If the data does not comply to the grammar, it does not comply with its security policy or it makes the overall data graph in the data space inconsistent, the data is rejected and the operation is rolled back.*
(*RDF*, *Data-Space-Handle*) := **get** (*Pattern*)	*Since the SDS can contain multiple data spaces, the simple* ***get*** *operation non-deterministically retrieves from one data space in the SDS data matching a pattern together with the reference to the source if data.* **get** *will only retrieve data from the SDS whose security policy complies with the security status of the requesting application or user.*
RDF := **get** (*Pattern*, *Data-Space-Handle*)	*The second version of* ***get*** *allows a Service Mediator to already address the data space where the pattern should be tested for a match.* **get** *will only retrieve data from the SDS whose security policy complies with the security status of the requesting application or user.*
subscribe (*Pattern, Callback*)	*As extension to the get operation the SDS support to* ***subscribe*** *for the occurrence of data in data spaces that match a given pattern. The callback is a reference to a computation within a Service Mediator which will be called by the SDS to notify the occurrence. A subscribe is denied if it potentially conflicts with the security policy of data*

	or the privacy policy of the user/node.
subscribe (*Pattern*, *Data-Space-Handle, Callback*)	*A mediator that is just interested in the occurrence of data in a specific data space can use the second variant of the* ***subscribe*** *operation. A subscribe is denied if it potentially conflicts with the security policy of data or the privacy policy of the user/node.*

2.4 Networking functionalities

One of the innovative aspects of SerWorks is the application to the networking context of concepts and techniques developed traditionally at the service level [SAC07a, SAC07b]. The overall objective is to enhance the system flexibility by allowing nodes/services to switch dynamically between different implementations of network-level functionalities. In this way, we enable services to build at run-time a sort of "service-tailored network protocol stack". This is consistent with the "Disappearing Network" concept, which looks at the network as a mere appendix of services that can be dynamically created and modified in order to match the service needs.
The following service-level concepts and techniques will be applied to network-level components:

- modular architecture: well-defined functional blocks with standardised APIs are identified; their interconnection provides the desired network-level functionalities;
- late binding: different implementation of single modules can be present in the system, and the control logic can decide at run-time which one has to be used for a specific service request;
- concepts and algorithms for dynamic composition: the composition graph related to network-level components can be modified at run-time according to the concepts and techniques developed in [BIONETS_D3.2.3].

At the same time, the following difference exists:

- compulsoriness: (some of) the network-level components are necessary in order to let the BIONETS system perform successfully and securely;
- locality: network and security-level modules cannot be composed across nodes but need to reside on the local node.

The BIONETS networking framework includes eight functional modules (components). Each module corresponds to a set of routines necessary for performing a given task. These modules are: (i) opportunistic communications, (ii) naming system, (iii) information filtering, (iv) data dissemination, (v) interoperability with IP networks, (vi) information gathering, (vii) secure communications and (viii) P2P cloud. Each of the components is represented by one (or multiple) service cells. Different service cells implementing one of the specific components can be present. Mediators are used for providing late binding and dynamic composition capabilities, as well as for enhancing the components' behaviour with autonomic functionalities. Some of the components are assumed to be static (e.g., naming system), i.e., we do not foresee the possibility of having different species thereof coexisting. (At the same time, this does not prevent the possibility for them to change over time.)
As anticipated, some of the components are mandatory, depending on the type of node considered and the role played in the overall system architecture. In particular, the following Table summarizes the set of mandatory components (denoted by a [M]) and optional components (denoted by a [O]). The mandatory components are those required to ensure fulfilment of the basic BIONETS networking functionalities [BIONETS_D112]. The presence of the optional components may respond to specific needs (e.g., securing T-to-U-Node communications for sensitive data) or represent performance enhancements (e.g., the possibility of performing information filtering on T-Nodes).

Type of Node	*Networking Framework Components*
T-Node	opportunistic communications [M] naming system [M] information filtering [O] secure communications [O]

Type of Node	***Networking Framework Components***
U-Node	opportunistic communications [M] naming system [M] information filtering [M] data dissemination [M] information gathering [M] secure communications [M] P2P cloud [M]
AP	opportunistic communications [M] naming system [M] interoperability with IP networks [M] information gathering [O] secure communications [M]

In the following, we detail the functionalities offered by each of the network-level module:

- **Opportunistic Communications.** BIONETS is based on localized interactions, based on the use of proximity wireless communications. Meetings among nodes are opportunistically exploited to diffuse data system-wide, according to a DTN-like store/carry/forward paradigm [Fall03]. The Opportunistic Communications module implements all functionalities necessary to perform such exchanges of data with peer nodes, including, e.g., node discovery, fragmentation/defragmentation. It interfaces directly with the underlying LLC level primitives.
- **Naming System.** BIONETS relies on *names* for identifying communicating devices. Names in BIONETS are intentional and are defined as a set of pairs <attribute,value>. Names are location-independent identifiers, i.e., they have global spatial scope and do not change as the node moves in the system. Names have limited temporal scope, i.e., they might change over time. All nodes in BIONETS have a name. Names are not unique. Names can be used for taking decisions concerning information/data forwarding, consistently with the *data-centric* nature of BIONETS networks [BIONETS_D111]. Nodes which are subject to trust and reputation systems (i.e., all U-Nodes and some classes of T-Nodes) possess a *unique static* identifier called *identity*. The identity of a node has global scope in space and time. It represents a fingerprint of the node, and it is expressed as a numerical value. We assume that node identities are hardwired in the nodes by the manufacturer. Identities are not used for taking forwarding decisions, and are not exposed to the network framework. Identities are exposed to trust and reputation services only. BIONETS encompasses also the use of identifiers with local scope in both space and time, which are termed addresses. An address is constituted by a numerical value, which can be associated to U-Nodes and to some classes of T-Nodes. Addresses are unique within a two-hop neighbourhood. Addresses are generated locally according to a random procedure, coupled with mechanisms for resolving collisions. More details are in [BIONETS_D112].
- **Information Filtering.** This module is responsible for managing the contextual data stored on the local memory based on their content as well as on the current context. This is necessary in order to ensure proper scalability with respect to environmentally-generated information, which can be gathered through T-Nodes. In particular, it includes techniques for dropping messages whose data content bears limited information on the current context (due, e.g., to the age of the message or the distance from the location where it was generated). Decisions are taken locally at each node based on the *metadata* describing the data type and attributes. Techniques for implementing information filtering have been introduced in [BIONETS_D111], [BIONETS_D112], and [BIONETS_D122].
- **Data Dissemination.** This module is responsible for ensuring the dissemination of data system-wide. The data dissemination in BIONETS relies on the movement of U-nodes and on a peer-to-peer data exchange. The U-nodes, after having read the information provided by the T-nodes, undertake the task of information dissemination. Information dissemination refers to

sharing the whole or part of the information with other U-nodes in the area, which may be interested in the information that is spread. The data dissemination module includes a set of policy that defines the behaviour of the node when getting in contact with another U-Node. Policies include also measures related to the reputation index of the node getting within communication range. Policies have to be engineered in such a way to ensure a proper trade-off between efficiency (efficient usage of resources available in the system) and utility (perceived by users as their ability to access information of interest). Some examples of policies are reported in [BIONETS_D122].

- **Interoperability with IP Networks.** This module is responsible for ensuring the ability of APs to *opportunistically* exploit the presence of infrastructure-based IP networks for enhancing the quality and range of offered services [BIONETS_D112]. Several usages are possible, including interworking from BIONETS to legacy networks (B2IP), from legacy networks to the BIONETS domain (IP2B) and interworking among BIONETS islands (B2IP2B). A combination of these techniques can be leveraged to decouple data retrieval from query dissemination [REVP2P].
- **Information Gathering.** This module is responsible for gathering environmental data from T-Nodes. It leverages the U-to-T-Node interface present in all U-Nodes [BIONETS_D112]. It can be used for both pulling data from T-Nodes (sending requests for a specific type of data) as well as for receiving data pushed by T-Nodes (applying an appropriate filter on the incoming type of message, according to the current set of interests).
- **Secure Communications.** This module is responsible to ensure secure communications between nodes on the network layer. On T-node level we identify different security classes which depend on the resources available to perform security relevant operations. Security modules to ensure secure communication on network level for U- and AP-nodes are mandatory. They provide functionalities required for authentication, confidentiality, and integrity of the network traffic and ensure the secure operation of, for example, the service life-cycle and the SDS.
- **P2P Cloud.** This module is responsible for the creation of a virtual overlay network, covering a BIONETS island and able to decouple the interaction/service framework and the characteristics of the underlying communication infrastructure for U-nodes interactions and the corresponding BIONETS communication protocols at network layer. The design of the P2P cloud functions is meant to extend the BIONETS network with features aiming at simplifying the implementation of the models envisaged for the interaction framework. The P2P cloud implements an overlay of peers and links; the functions of overlay peers are "hosted" on the U-nodes, while the virtual links are based on the underlying communication primitives. Disconnected islands of U-nodes generate separated overlays. The peers are containers for retrieving (in a broad sense) the entities of service framework, such as services, service descriptions, data/information, etc. Such overlay is meant to apply to situations in which the level of dynamicity present in the system is low (e.g., colleagues sharing an office).

BIONETS networking framework is based on the use of messages as fundamental data units (see App. A, [BIONETS_D112], and [BIONETS_D122]). The framework is meant to implement a pure data-centric system, in which the gathering, dissemination and management of data is based on the matching between (i) the data content, as defined in the metadata describing the message payload (ii) a set of interests, expressed as a set of policies according to which communications have to be performed. Policies are derived on the basis of both the user's profile (e.g., kind of information the user is interested in etc.) as well as running service needs (e.g., contextual information needed to enhance the current service). The latter point may include restrictions on the set of hosts with which interactions are allowed (including, e.g., support of given encryption mechanisms, reputation level etc.). Policies are, in general, described using the same <attribute,type> mechanism used for metadata description.

3. Interactions among Components

3.1 Interfaces

The following figure introduces the main interfaces among the frameworks, which are meant to enable the interactions among the components, deployed either on a single node or distributed across several nodes.

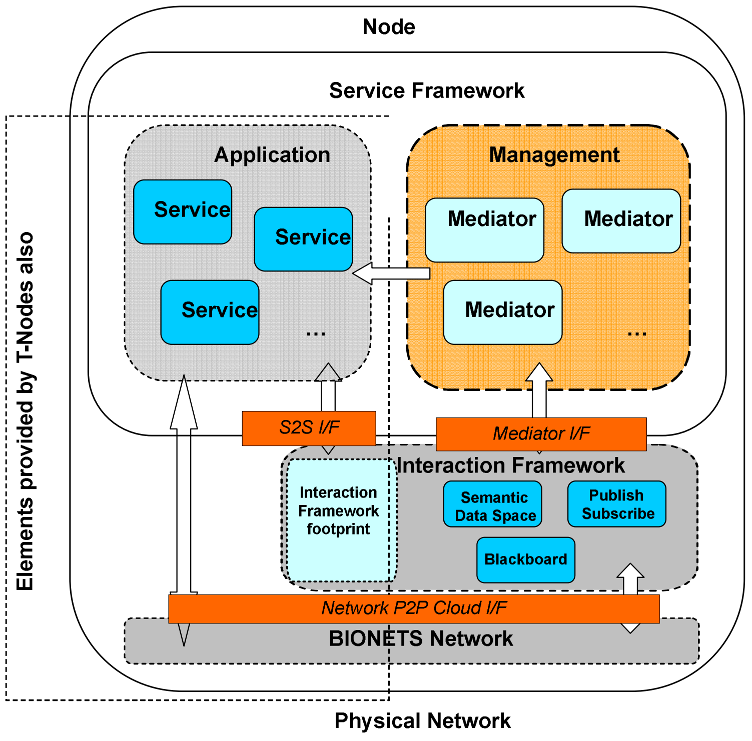


Figure 9: Main interactions interfaces of SerWorks

The interfaces are structured as a set of interaction primitives offered as APIs.
In order to support rich interaction models (e.g., asynchronous interactions, reception of information in "push" mode), the primitives of the interfaces offered by a framework are grouped in different APIs:

- "direct" APIs, grouping the operations for allowing an entity to request the capabilities of the framework; these operations are implemented by the framework;
- "call-back" APIs, grouping a set of operations invoked by the components of the framework to asynchronously provide information, events, results; these operations have to be implemented by the entities (e.g., services, mediators) which use the framework capabilities

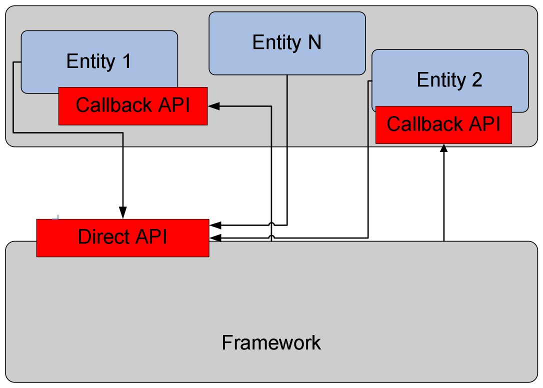


Figure 10: Interface's APIs: direct APIs and callback APIs

The interface offered by the interaction framework to the service framework is separated in two parts:

- **interface for direct service-to-service (or S2S) interactions**: these are used by the service cells and/or the service individuals to perform interactions (e.g., service invocations, event notification), and data searches;
- **interface for mediator interactions**: these are used by the mediators in order to share and/or exchange information relevant for service declaration, selection, etc. (e.g., announce service definitions, publish fragments of ontologies).

These interfaces, along with the associated primitives, are aligned with the interaction models described in [BIONETS_D3.2.1].

Interface for mediator interactions are based on CRUD primitives extended with a primitive for propagating notifications in a "push" mode, in order to handle publish/subscribe/notify model:

- `create(XML-document)` → `Result;`
- `read(XML-query)` → `XML-document;`
- `update(XML-query, XML-path, XML- fragment)` → `Result;`
- `delete(XML-query)` → `Result;`
- `push(XML-document)` (this primitive is part of the callbackAPI)

where:

- `XML-document` is a XML document or a portion of it;
- `XML-fragment` is a portion of an XML document;
- `XML-Query` is a statement of a language for querying XML documents (examples are: XQL, the XML Query Language defined by W3C; SPARQL, a query language developed primarily to query RDF graphs); it is used to select the documents to be read, updated, or deleted;
- `XML-Path` is a statement of a language for addressing parts of an XML document (an example is XPath); it is used to determine the part of a XML document to be modified;
- `Result` reports the result of an operation, e.g., the success or an exception.

The parameters of the primitives are assumed to be XML-based, as this interface is mainly used to access the Semantic Data Space (SDS), and the standard languages that can be used to code the semantic information stored in the SDS (e.g., service descriptions, ontologies, etc.) are XML-based (e.g., OWL, OWL-S, RDF). In fact, XML-documents could be used to code semantics information (e.g., RDF data), in order to build a Semantic Data Space.

These primitives can be used to implement several interaction models. For instance, the publish/subscribe/notify model relies on these primitives in the following way:

- `publish` (of an information or a topic): it is implemented as a `create`;
- `subscribe` (to a topic of information): it is implemented as a `create`; the created data contains a reference to the call-back interface used for notifying the information;
- `notify` (of a subscribed information): it is implemented through a `push` primitive.

`XML-document` parameters should be structured in order to include the relevant information for the subscription and publication (e.g., the identifier of the subscribing entity, the reference of the (call back) push primitive, etc.).

These primitives can be used to access and manage a Semantic Data Space containing, for instance, data related to Service Descriptions. `XML-document` and `XML-fragment` parameters could be, therefore, structured in order to handle the relevant information such as:

- Service Description: it includes Bionets service cell/identity identifiers, and the description of functional and non-functional features; it is an (XML) tuple, including RDF or OWL fragments/documents, defined in D3.2.4.

```
<service:Service rdf:ID="ServiceID">
  <service:supports>
    <grounding:……/>
    <process  ……..../>
  </service:supports>
  <service:presents>
  <profile:Profile rdf:ID="Service_Profile">
</service:Service>
```

- Effect/gain/goal description: they are represented as RDF triple, in order to allow mediators the search or selection of services based on the effect description of the service

```
<process:hasEffect rdf:resource="&effect-c; …./>
```

Interface for direct service-to-service interactions provide primitives for implementing different communication patterns among services (e.g., point-to-point, or point-to-multipoint, epidemic information dissemination, epidemic information retrieval).

Primitives for service invocation (implementing RPC-like interactions: these are analogous to BPEL primitives for web service invocations, at both client and server side; an example of usage is reported in Figure 11):

- `service-request(from-service, service-reference, request) → response`:
  it performs an rpc-like invocation to a service; in case of one-way request (i.e., a request without an answer) the rpc primitive immediately returns; its asynchronous/not-blocking version is composed of two primitives, one for the request, the other, provided by the callbackAPI, for waiting the response:
  - `request-message(from-service, to-service,message) → rpc-id`;
  - `response-message(rpc-id, message)`.
- `service-registration(service-id, service-type)`:
  it is used by a service cell/individual to inform the Interaction Framework on the type of service is able to offer;
- `invocation-request(from-service, rpc-id, request-message)`:
  it is a *call-back* operation offered by a service cell/individual to wait for requests of service invocations, and activate a service execution;
- `invocation-response(rpc-id, response-message)` :
  it is used by a service cell/individual to send a response containing the results of a service invocation.

Primitives for (epidemic) information retrieval (concerning data required by services and offered/elaborated by services):

- `search-data(query-request) → query-answer`:
  it is used by a service to start a search of data among the U-nodes; its asynchronous/not-blocking version is composed of two primitives, one for the request, the other, provided by the callbackAPI, for waiting the answer:
  - `search-data-request(query) → query-id`;
  - `search-data-response(query-id, query-answer)`;
- `query-registration(query-type)`:
  it is used by a service cell/individual to inform the Interaction Framework on the class of queries that the service is able to solve;
- `query-resolution(query-to-be-solved) → query-answer`:
  it is a *call-back* operation offered by a service cell/individual to allow invoking a request to resolve a query.

Primitives for (epidemic) information dissemination/notification:

- `event-subscription(topic):`
  it is used by a service cell/individual to declare its interest in receiving events/information on a given topic;
- `event-publishing(topic, message):`
  it is used by a service cell/individual to send an event of a given topic;
- `event-notification(topic, message):`
  it is a *call-back* operation offered by a service cell/individual to receive events/information of a given topic.

The **network P2P cloud interface** includes a set of interaction primitives that are offered to the interaction framework by the network framework, for communication among nodes and data storage. The primitives described in the following table are a refinement of the ones described in [SAC07a]. These primitives can, also, be used by service components, in order to perform transfer of "bulk" data.

Interaction Primitives	Description
`publish (Data, Properties)`	Request to publish/announce/propagate information to the interested entities. `Data` is the information to be published (e.g., a file, a document, a service description). `Properties` is a set of characteristics (e.g., a key, a set of keywords/topics, a property list, semantic description) through which it is possible to select information that are relevant for an entity through a filter.
`deprecate(Filter)`	Request to deprecate a (set of) previously published data. `Filter` is used to discriminate the data to be deprecated, by considering their Properties.
`search(Query, Query-id)`	Request to start a search of a data fulfilling the "filter" represented by `Query. Query-id` is a unique identifier (e.g., including the identifier of the requesting entity/node) that the will be used by the entity to retrieve the searched information.
`send(Peer, Message)`	Request to send a `Message`, i.e., a command, the answer of a command, to an entity (e.g., a node, a service). `Peer` is a "filter" that identifies the entities which are the destination of `Message`: it could identify multiple entities, e.g., to "broadcast" a message. Message could contain an identifier that could be used to identify a possible response.
`receive(Communication-Primitive-Message) →  boolean`	Call back primitive, for waiting for/receiving a `Communication-Primitive-Message` to be elaborated by the entity; this primitive returns a boolean to indicate whether the message is relevant or not for the entity (i.e., the Interaction Framework module instantiated on a node)
`retrieve-request(Data-id, Mode, Request-id)`	Request to retrieve the information associated to `Data-id` (e.g., it could be a Query-id). `Mode` indicates whether the information must be deleted from the permanent storage or not. `Request-id` is a unique identifier used to associate the message returning the result.
`retrieve-result(Request-id, Result)`	Call back primitive, for waiting for/receiving the result of a retrieve-request. `Request-id` is the unique identifier used to discriminate the relevant result. `Result` is either the retrieved data or an exception indication

These primitives could have different "implementations", depending on the whether P2P cloud includes permanent storage capabilities or not. Examples are the network overlay functions as described in [BIONETS_D122] or the Reverse P2P middleware described in [DePellegrini08].

3.2 Flow of interactions

The following figure illustrates the sequence of primitive invocation of the interface for direct service-to-service interactions for the execution of an invocation of the service2 on node2 performed by service1 on node1. The flow described a simple RPC-like service interaction, with asynchronous return of the result to deal with loosely coupled communications [BIONETS_D321].

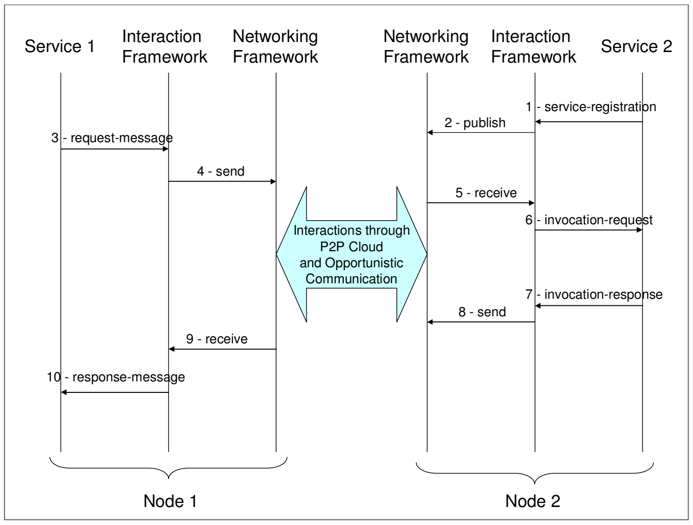


Figure 11: Service-to-service interactions

The following figure describes the interaction performed by mediators' interface, based on CRUD operations on a Semantic Data Space storing (semantic) service descriptions. The figure presents the interactions performed by a mediator (Mediator 1) for publishing a service description and the ones performed by another mediator (Mediator 2) for selecting a service fulfilling some requirements.

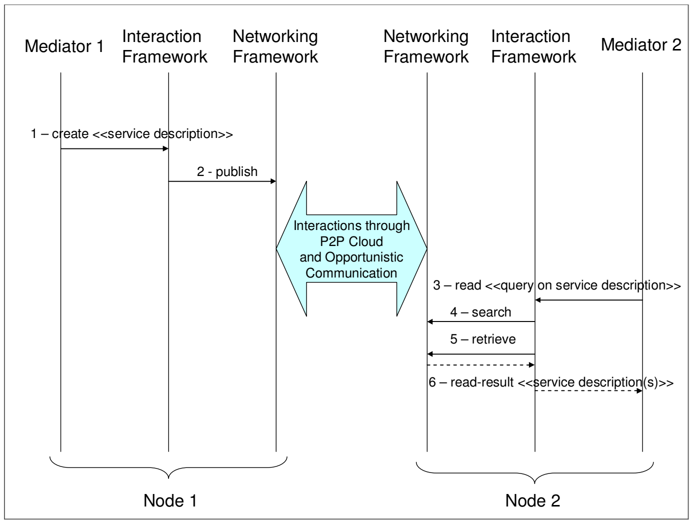

Mediator 1
Interaction Framework
Networking Framework
Networking Framework
Interaction Framework
Mediator 2
1 – create <<service description>>
2 - publish
Interactions through P2P Cloud and Opportunistic Communication
3 – read <<query on service description>>
4 – search
5 – retrieve
6 – read-result <<service description(s)>>
Node 1
Node 2

Figure 12: Mediators' interactions, mediated through the Semantic Data Space

Both pictures focus on the interactions concerning the Service Framework and the Interaction Framework, and abstract from the ones performed by the networking framework, in particular by Network P2P Cloud and Opportunistic Communication which are in charge to handle node communication at networking layer. A detailed description of them can be found in [DePellegrini08].

3.3 Local vs global interactions

The following table summarizes the mechanisms for the interactions of service components/mediators internally and externally to a node. This table mainly relates to service framework components.

	Local		*Remote*	
	Service Components	Mediator	Service Components	Mediator
Service Components	Control: Interaction Framework Data: direct interactions through local interfaces	Direct interaction through local interfaces	Control: Interaction Framework (through S2S i/f)+ Networking Framework Data: Networking Framework	not allowed
Mediator	Direct interaction through local interfaces	Interaction Framework with SDS or Direct interaction through local interfaces	not allowed	Interaction Framework with/ SDS + Networking Framework

Control interactions among service components are always performed through the Interaction Framework, in order to guarantee transparency with respect to the service components location. Service components could exploit the different features and interaction models provided by Interaction Framework for interacting with other service components (service request, data, search, shared data space, etc.), abstracting from the networking complexity.
Transfer of "bulk" data among service components can be performed, on the other hand, by accessing directly the networking capabilities.
Service components interact with local mediators, by means of local interfaces, supported by the node operating system (e.g., Java interfaces, in case of JavaVM). Interactions between mediators and service components deployed on different nodes are not allowed; this constraint is motivated by the need to enforce the control, performed by mediators, on the usage of the services deployed a node; moreover, this solution would hide to service components distribution complexity (which is completely handled by mediators). Remote interactions of service components are always mediated by local mediators: for instance, when a service component S_a has to perform a binding with another service component S_b, S_a has to contact a mediator M_a located on the same node; M_a performs the requests of S_a (through the Interaction Framework features) by interacting with a mediator M_b located on the same node of service component S_b.
Mediator interactions are performed either through the Interaction Framework, possibly according to the interaction model supporting the Semantic Data Space or, if they are deployed on the same node, through local interfaces. Semantic Data Space is used by Mediators to store/retrieve/reason on information related to service components.
Interactions between components of the Interaction Framework deployed on different nodes are performed through the capabilities provided by the Networking Framework; while interactions among components deployed on the same node are performed by means of the local interfaces, supported by the node operating system.
Finally, in order to enforce security and networking policies, services related to security functions and network level services can be accessed only from components deployed on the same node. Communication among nodes is enabled by the interactions among Opportunistic Communication modules, performed through proximity wireless communications.

4. Conclusions

In this deliverable, we have introduced the first version of the SerWorks architecture. SerWorks enables a unified representation of network- and service-level BIONETS components, based on three main features: modularity, reusability and late binding. The main benefit foreseen for SerWorks-type architectures relates to the inherent flexibility added to the system in terms of network-level functionalities, which are conventionally static and cannot be changed at run-time in response to either varying operating conditions as well as specific needs of the running services.

This document is meant to set the baseline for the algorithms and methods to be developed in WP1.2 and WP3.2. The second iteration of the SerWorks architecture, expected at M42, will profit from the understanding gained through prototyping and experimental activities carried out within WP5.

References

[vanRenesse95] R. van Renesse, K. P. Birman, R. Friedman, M. Hayden, and D. A. Karr. "A framework for protocol composition in HORUS." In *Symposium on Principles of Distributed Computing*, pages 80–89, 1995.

[Braden02] R. Braden, T. Faber, and M. Handley. "From protocol stack to protocol heap — role-based architecture," 2002. *First Workshop on Hot Topics in Networks (HotNets-I).*

[Carreras07] I. Carreras, I. Chlamtac, F. De Pellegrini, and D. Miorandi. "Bionets: Bio-inspired networking for pervasive communication environments." *IEEE Trans. Veh. Tech.*, 56:218–229, Jan. 2007.

[Carreras07b] I. Carreras, D. Tacconi, D. Miorandi, "Data-Centric Information Dissemination in Opportunistic Environments", in Proc. of IEEE MASS (Demo Session), Oct. 2007.

[MN06] R. Moskowits, P.Nikander, Host Identity Protocol (HIP) Architecture, IETF RFC 4423, May 2006.

[Linner07] David Linner, Heiko Pfeffer, and Stephan Steglich. "A genetic algorithm for the adaptation of service compositions," in *Proc. of BIONETICS, SAC Workshop*, Budapest, Dec. 2007.

[BIONETS_D321] H. Pfeffer, D. Linner, S. Steglich and I. Radusch (Eds.), "Specification of Service Life-Cycle", BIONETS D3.2.1, Feb. 2008.

[BIONETS_D324] J. Lathi (Ed.), "Advanced Service Life-Cycle and Integration", BIONETS D3.2.4, Jul. 08.

[BIONETS_D54] I. Carreras (Ed), "Pervasive ubiquitous peer-to-peer context-aware application", BIONETS D5.4, Aug. 08.

[SAC07a] F. De Pellegrini, D. Miorandi, D. Linner, L. Bacszardi and C. Moiso, "BIONETS: from Networks to SerWorks" in *Proc. of BIONETICS, SAC Workshop*, Budapest, Dec. 2007.

[SAC07b] D. Bliefernicht, D. Schreckling, "Highly Adaptive Cryptographic Suites for Autonomics WNSs", in *Proc. of BIONETICS, SAC Workshop*, Budapest, Dec. 2007.

[BIONETS_D323] F. Baude and L. Henrio (Eds.), "Graph-based Service Individual specification: Creation and Representation", BIONETS D3.2.3, Jan. 2008.

[Fall03] K. Fall. "A delay-tolerant network architecture for challenged Internets". In *Proc. of ACM SIGCOMM*, Karlsruhe, DE, 2003.

[BIONETS_D122] D. Raz and F. De Pellegrini (Eds.), "Disappearing Network Autonomic Operation and Evolution", BIONETS D1.2.2, Jul. 2007.

[BIONETS_D112] D. Miorandi (Ed.), "Architecture, Scenarios, and Requirements refinements", BIONETS D1.1.2, Aug. 2007.

[BIONETS_D111] D. Miorandi (Ed.), "Application scenario analysis, network architecture requirements and high-level specification", BIONETS D1.1.1, Jul. 2006.

[DePellegrini08] F. De Pellegrini, I. Carreras, D. Miorandi and C. Moiso, "R-P2P: a Data Centric DTN Middleware with Interconnected Throwboxes", to appear in *Proc. of Autonomics 2008.*

[Kephart03] Jeffrey O. Kephart, David M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, n. 1, pag. 41-50, 2003.

[Tschudin91] Christian F. Tschudin, "Flexible Protocol Stacks," in *Proc. of ACM SIGCOMM* 1991: 197-205

[OMalley92] Sean W. O'Malley and Larry L. Peterson , "A dynamic network architecture", *ACM Transactions on Computer Systems*, vol. 10, n. 2, pag. 110-143, May 1992.

[Frantti08] T. Frantti, J. Huusko, R. Savola and V.Könönen, "Architecture Requirements of the Future Internet, " in *Proceedings of ACM 4th Mobile Multimedia Communications Conference*, July, 2008.

Appendix A: Terminology

- **T-Nodes**. T-Nodes are simple, inexpensive devices with sensing/identifying and basic communications capabilities. T-Nodes act as an interface with the environment and are used to gather contextual information which is utilized by the U-Nodes to provide context-awareness. T-Nodes do not communicate among themselves but are just read by U-Nodes in proximity. They present minimal requirements in terms of processing/storage/communications.
- **U-Nodes**. U-Nodes are complex, powerful electronic devices with computing and communication capabilities. PDAs, laptops and smartphones represent examples of a U-Node. U-Nodes are typically carried around by users and therefore are inherently mobile. Their mobility is exploited, in BIONETS, to provide system-wide diffusion of messages. U-Nodes host services. They interact with the environment through T-Nodes, from which they gather the contextual information necessary to provide the users with services enhanced by context-aware features. U-Nodes may communicate among themselves to exchange information, such as environmental data or service-specific code (in order to enable service evolution).
- **Access Points (APs)**. Access Points are complex powerful devices that may be used for (i) accessing IP-based services by the BIONETS networks (ii) collecting environmental data (through BIONETS system) from a remote IP service (iii) providing IP shortcuts among disconnected BIONETS islands. APs are envisioned to act as *proxies* between BIONETS networks and IP networks.
- **Service cell (SC)**: an atomic logic entity, which may provide knowledge, content, or functionality to other services and users.
- **Service individual (SI)**: a SI is a composition of Service Cells. Service Individuals may also be composed of other Service Individuals in a recursive way. Service Individuals are expressed with a graph representation [BIONETS_D323], wherein nodes represent types of Services (the elements of the composition); edges (links) represent the interactions (communication) between the Services.
- **Mediators**: logic entities performing control functions related to the operations of SCs and SIs. In particular, mediators implement the autonomic functionalities encompassed by the BIONETS service lifecycle.
- **Messages**. Communications in BIONETS are based on the exchange of messages. Messages are service data unit, i.e., encapsulation of data items meaningful to a service. In general, messages will be much larger than standard IP packets. (This is because single IP packets usually do not expose meaningful data to the service layer.) Messages will consist of a payload (or content) and metadata (expressed as a set of <attribute,value> pairs) carrying the necessary information for the node to decide which operations should be undertaken. Communications in BIONETS are asynchronous and connectionless. Messages are treated as datagrams, and the whole system can be thought as a message-switching engine.
- **Opportunistic Forwarding**: mechanism for diffusing information in a highly partitioned network, based on the exploitation of "contact opportunities" between nodes in the system. Opportunistic forwarding is based on localized interactions only and exploits mobility of the nodes to ensure network-wide diffusion of messages.
- **Information Filtering**: mechanism for limiting the diffusion of data messages with low information content. This is related to the fact that, in most context-aware applications, context-related data looses its usefulness when being far (in both space and time dimensions) from the originating context. Information filtering is an essential building block of data management in BIONETS systems.
- **Identifier**: an identifier is a finite sequence of symbols of a given alphabet, used to identify an entity within a set of entities. Identifiers have a scope (in space and time) which determines the domain within which they can be used for identifying entities.
- **Name**: location-independent (i.e., with global spatial scope) identifier of a logical BIONETS entity. A name in BIONETS is constituted by a set of pairs < attribute, value

>. Names in BIONETS are intentional, i.e., they can be used by services and applications to specify what they are looking for. Names in BIONETS are dynamic, i.e., they may change over time (equivalently: they have a limited scope in time).

- **Address**: location-dependent identifier (i.e., with local spatial scope) of a logical BIONETS entity (in this deliverable: node). Addresses can be used for identification purposes only within a connectivity island. Addresses have a limited scope in time. Addresses can be generated on-the-fly according to a random procedure by each U-Node and by some classes of T-Nodes. Addresses are numeric sequences and can be dynamically bound to names. Addresses may be used for performing one-hop point-to-point communications among BIONETS entities.
- **Identity**: globally unique identifier associated to each U-Node (and to some classes of T-Nodes). An identity has global scope in both time and space. Being location-independent, an identity is technically a name (even if we will use the two terms separately to avoid confusion).
- **Security principal**: any node in the BIONETS network architecture that can be authenticated.

Appendix B: Interfaces for Service Discovery, Adaptation and Migration

In the following figures we report some of the main Mediator functionalities, such as service discovery and adaptation, and required interfaces in more detail. Different Mediator types and their functionalities are described in more detail in Sec. 2.2.
Fig.13 illustrates the basic required interfaces for service discovery mechanism for U-node including the T-node interfaces.

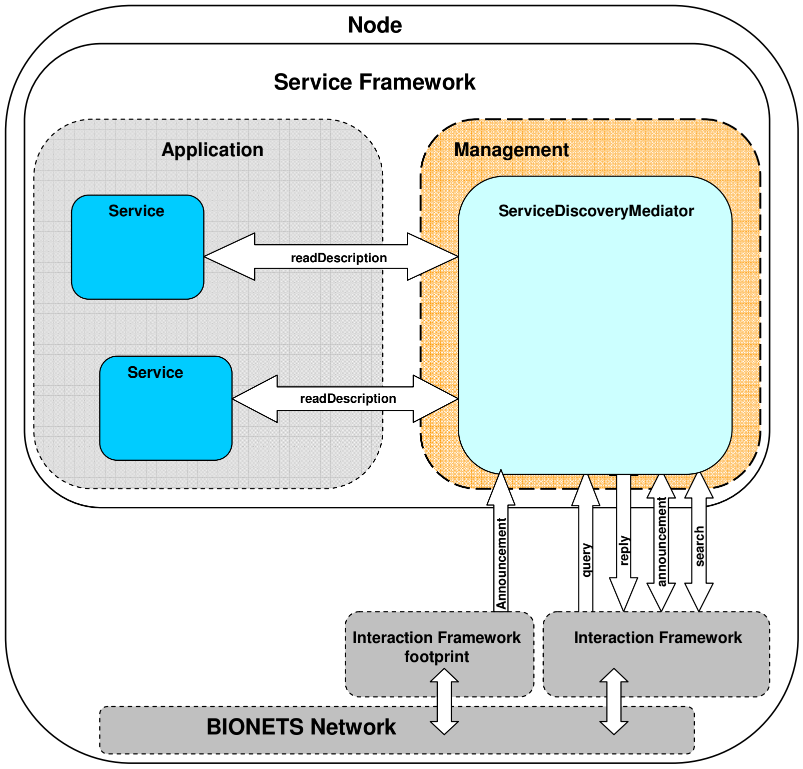


Figure 13 Service discovery operations

The main primitives include the service announcements, search, query and reply through the Interaction framework and service description from the service cells and individuals. The Service adaptation, graphically depicted in Fig. 14, vice versa requires additional primitives for context data such as bandwidth, battery consumption, CPU load, adaptation, monitoring and execution.

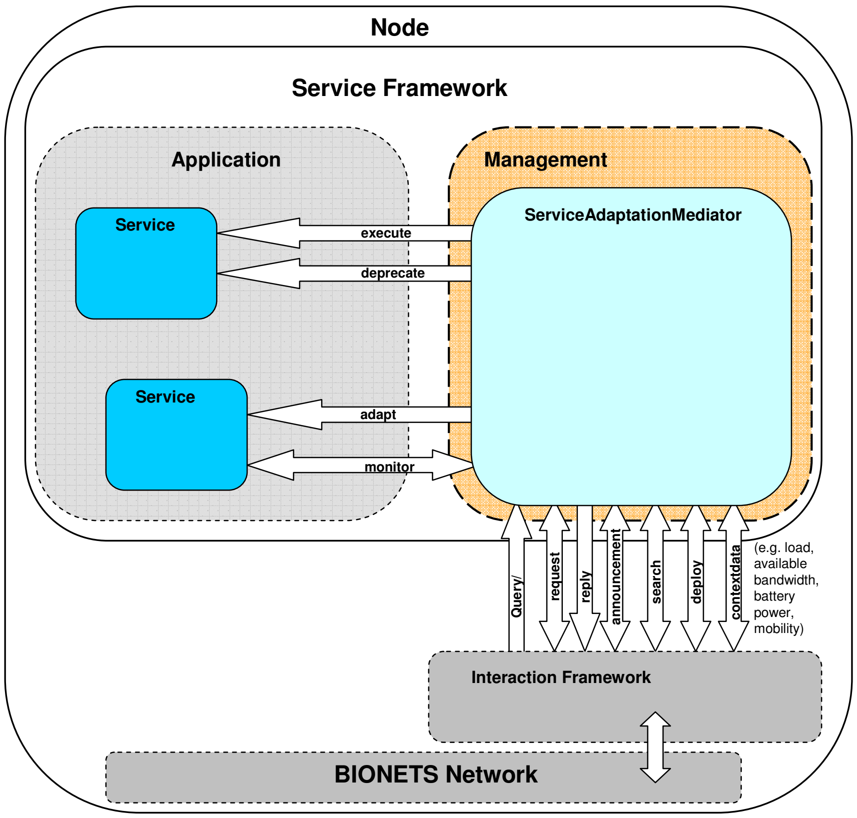


Figure 14. Service adaptation.

In Fig. 15 we have illustrated the secure service migration through the BIONETS system with the additional user interaction functionalities. In this case, the migration Mediator requires feedback from the network (i.e. the network state information, such as connectivity, link status, bandwidth etc.), user preferences, service requirements and node capacity information in order to make a decision if and when the migration is possible. The network state information needs to be then delivered by the P2P cloud component through the Interaction Framework to the Mediator. After the mediator decided to transfer the service from one node to another, the actual transmission is performed through the overlay. In the case of legacy IP networks and IPsec or e.g. using Host Identity Protocol (HIP) [MN06] the secure transmission between authenticated nodes can be performed in the peer-to-peer overlay using, e.g. HIP tunnelling or key-based routing by binding the service identifiers to network naming and identifier components. The required security functionalities for the network layer, i.e. authentication protocols, integrity protection, encryption etc., have to be provided by the Secure Communications component.

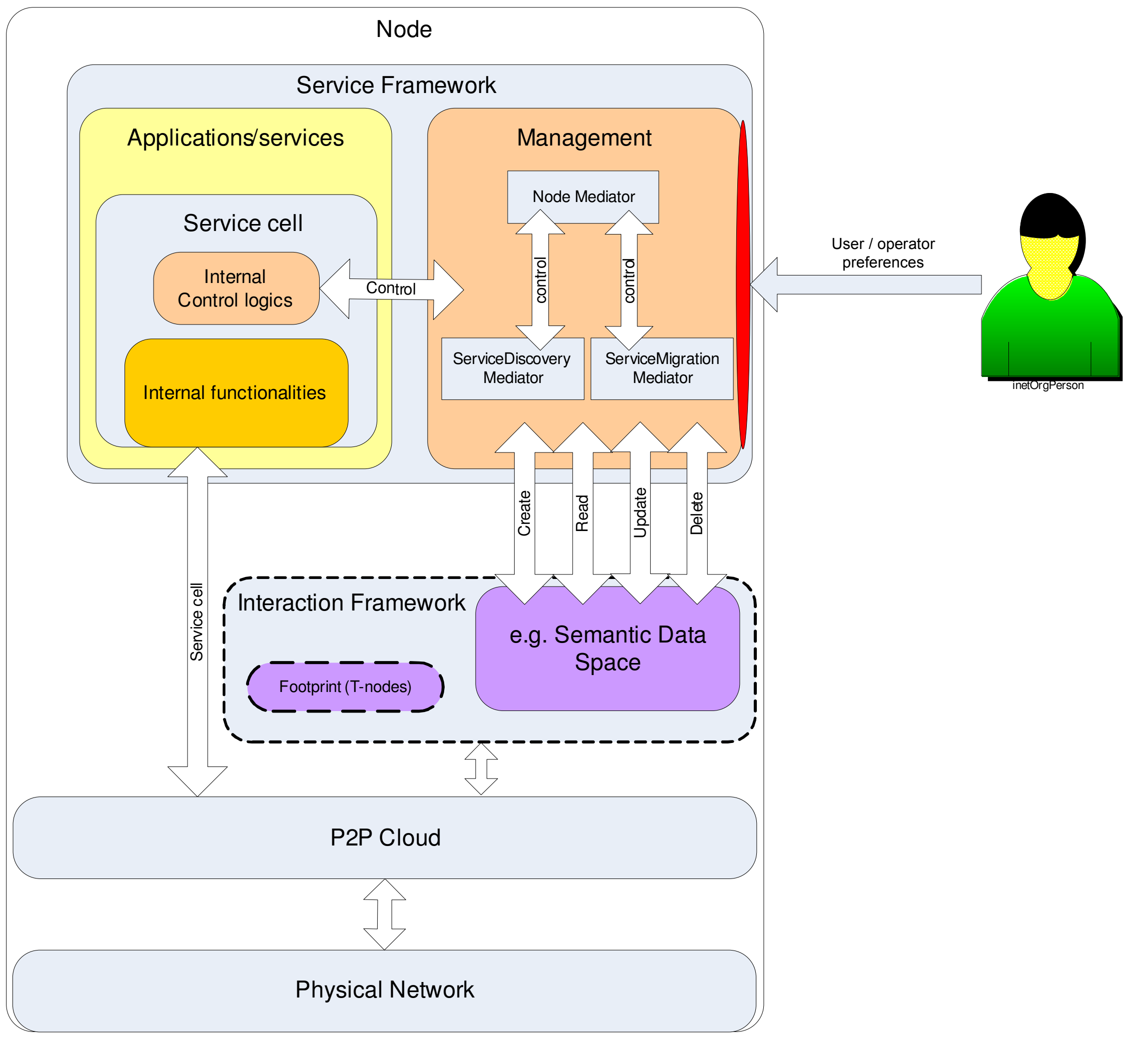


Figure 15.Service migration with P2P Cloud.

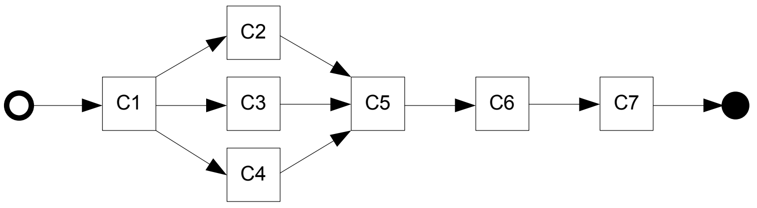
.

Appendix C: An Example: the Unsolvable Quiz Application

The Unsolvable Quiz is a self-evolving quiz game for mobile phones. The Unsolvable Quiz, presented at the Y-2 review meeting, was designed by the consortium as a first attempt to integrate network- and service-level solutions devised by the BIONETS consortium partners.
A run of the quiz presents a predefined number of questions to the user. Selection, number and order of questions are supposed to be significant for the difficulty of a quiz. During a run of the quiz the only relevant information is whether the user answers correctly to a question or not. The relation of correctly answered questions to the overall number of questions in the quiz is utilized as indicator for the fitness of a quiz. Higher fitness is associated to more difficult questions. If the fitness of the quiz falls below a predefined threshold an evolutionary step is performed to change selection, number and order of questions. The evolutionary step builds on tools and techniques inherited from genetic algorithms. The resulting candidate quiz with the highest probability to be unsolvable is selected from the generated offspring as new quiz.
At the networking level, the Unsolvable Quiz makes use of the U-Hopper middleware [Carreras07b, BIONETS_D54], running on J2ME-enabled mobile phones. U-Hopper is used for exchanging information on the quizzes in use by the devices and their respective fitness levels. The filters on the information to be dispatched are driven by the application itself, in line with the basic SerWorks approach.

Quiz Structure and Execution

The quiz is organized around questions. Each question is implemented as a Service Cell. Each question has one correct answer and belongs to one category. Examples for categories are European history, computer science, or biology. Moreover, each question is assigned a weight (between 0 and 1) that indicates its fitness level, defined as difficulty level. The difficulty level of a question is estimated as the fraction of users who answered incorrectly to it. Finally, a service cell is assigned a security policy which is based on the information it carries. Depending on the security credentials a device can present, the quiz can adapt its questionnaire. For example, a user may decide to involve all its documents on the BIONETS project in the quiz. However, this should only happen if the user or users participating in the quiz can prove that they are partners (or: evaluators, project officers) of the project.
The order of questions for a run of the quiz is determined by a directed graph. This graph is in the following, with a slight abuse of notation, referred to as "questionnaire". A questionnaire can be understood as an instance of a Service Individual, though the data-flow graph is missing. A branch in the questionnaire means that one of the options is selected in a non-deterministic fashion. The questionnaire refers to question categories. All questions available on one device are held in a pool comparable to the service container in the BIONETS Service Framework. For a run of the quiz the questionnaire is interpreted step by step. For each category one question is selected from the pool and presented to the user. The selection depends on the weight that indicates the difficulty of the question (e.g. most difficult question first) and its security policy. If the question is answered correctly the interpreter decreases the weight accordingly. The weight of an incorrectly answered question is increased.



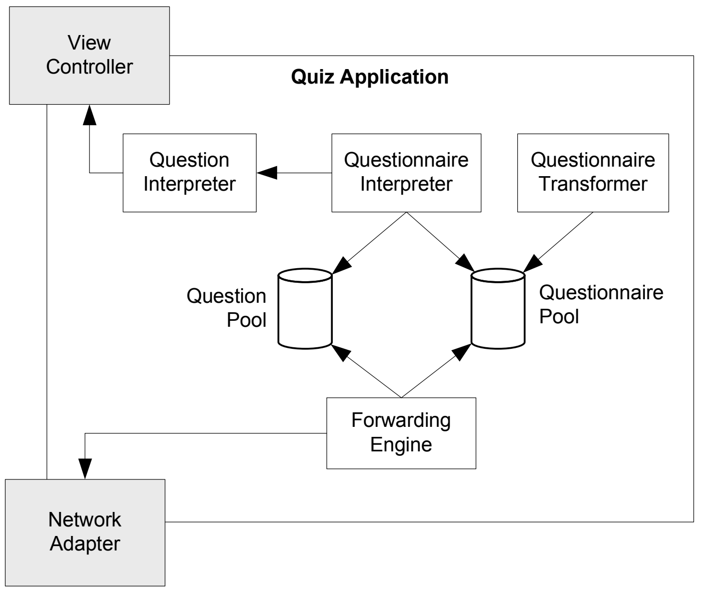


Figure 16: Structure of the questionnaire of a quiz

In the current implementation, questions are represented as small HECL scripts.[1] Presenting a question to the user means to execute the script. The script terminates either with true or false, indicating whether the presented question was answered correctly or not. The presentation of the question to the user is handled by the script itself. The script is therefore given full access to the presentation capabilities of the mobile application. For instance, a question script could present a question text along with four possible answers, while the user needs to click the correct one. Another example is a question script that presents a photo on the display and asks for the name of the person in this photo, while the user needs to enter the right answer letter by letter into a free form field.
Like questions, multiple questionnaires are held on each device, but there is one primary questionnaire which is always interpreted when the quiz execution is requested. The other questionnaires will be used as "genetic" material for the evolution of the quiz.

Online Evolution of Quiz

The quiz evolution aims at keeping the quiz unsolvable, so that it is able to challenge the user and keep its interest. The fitness function is used as an indicator of the difficulty level of the current combination (in terms of questions and order). Conventional GA techniques (crossover, mutation and natural selection) are used for enhancing the difficulty of the questionnaire. An unfit questionnaire can be improved through reordering question categories in the questionnaire, removing categories or introducing new ones. The improvement is supposed to be a non-linear optimization problem, since the fitness of question categories is considered and the user continuously learns with each run of the quiz. Hence, question weight and individual difficulty of a category for a particular user are just

[1] www.hecl.org

indicators. To enforce reasonable results for the evolution process, heuristics are applied that constrain the structure of questionnaires, such as the limitation of a single questionnaire's size.
The fitness of the offspring (crossed and mutated) is estimated based on earlier runs of the quiz and the overall difficulty of available questions for each referenced category. The questionnaire with the most promising results in the fitness estimation is selected as primary questionnaire. The following runs of the quiz and the implicated user feedback are used to get an impression about the real fitness of the selected questionnaire. Other questionnaire candidates from the offspring with a good or satisfactory result in the fitness estimation are added to the pool for later evolution steps.
An evolution steps is performed when the fitness of the primary questionnaire falls below a predefined threshold. Additionally, an evolution step can be performed when new questions and questionnaires arrive on a node.

Networking Framework

The networking framework for the Unsolvable Quiz is implemented by means of the U-Hopper platform, underpinning the Disappearing Network concepts, which include a pure data-centric architecture, an interest-driven data dissemination scheme and the use of an intentional naming system. The operations of such middleware are described in details in [BIONETS_D534]. Among the eight components of the BIONETS networking framework, the following ones are supported:

- naming system;
- data dissemination;
- opportunistic communications;
- information filtering;
- information gathering (not used in the demo).

In general, the demo does not leverage all the potential of the BIONETS networking framework, but exploits the opportunistic networking capabilities provided in order to support evolutionary mechanisms, by enabling the secure (if one of the service cells requires it) exchange of genotypes (and related fitness values) among nodes in the system. In particular, the dissemination of information in the system is driven by the application itself, through the definition of a set of filtering policies on the metadata accompanying the message itself. The resulting architecture of the Unsolvable Quiz prototype is depicted in Fig. 17.

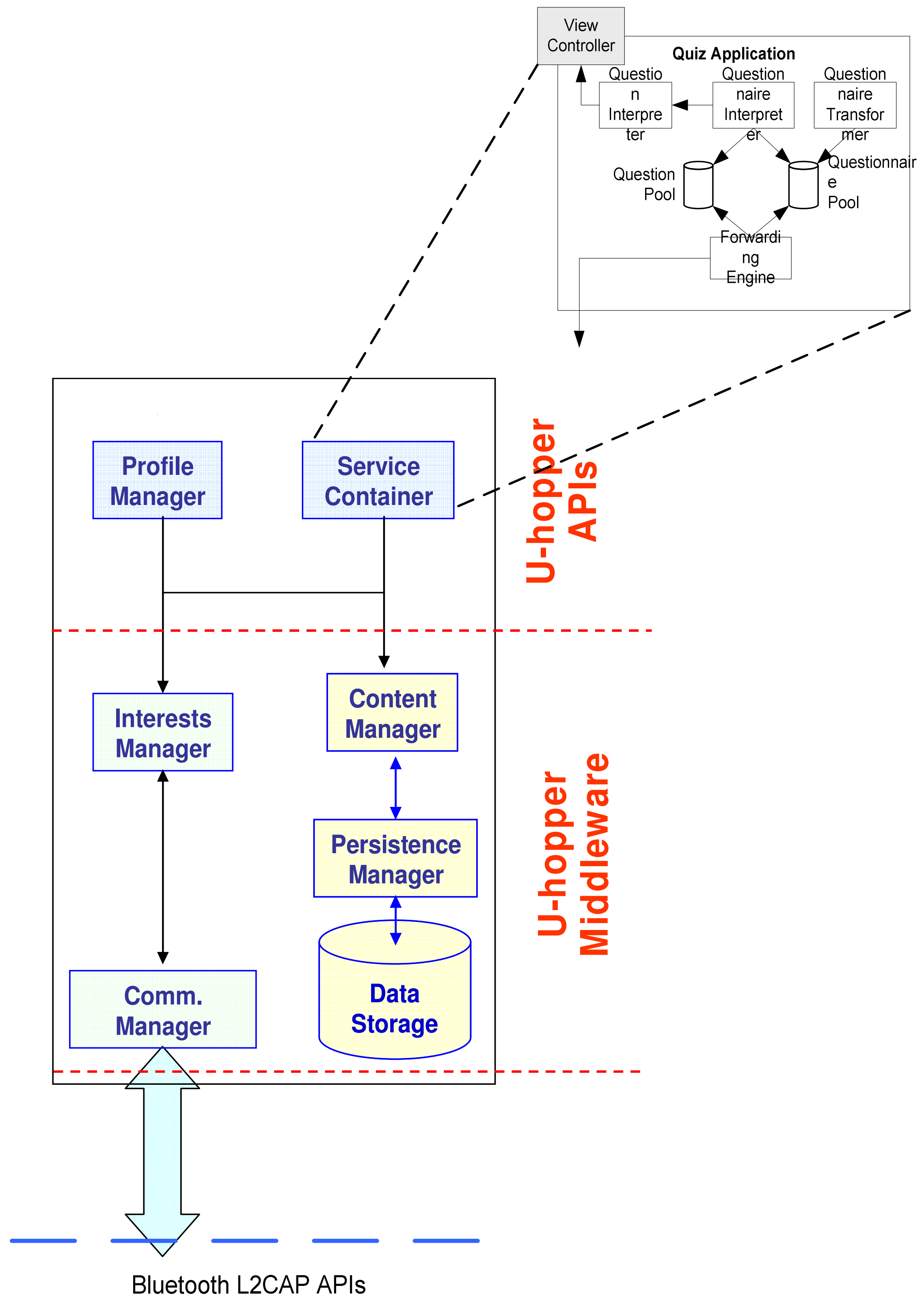


Figure 17. Architecture of the Unsolvable Quiz demo.