DOI 10.12694/scpe.v16i3.1100
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 3, pp. 249–269. http://www.scpe.org

OVERLAY SERVICE COMPUTING - MODULAR AND RECONFIGURABLE COLLECTIVE ADAPTIVE SYSTEMS

EVANGELOS POURNARAS *

Abstract. Distributed software systems that determine virtual communication structures on top of physical networks, the overlay networks, are a well-established approach to build various applications of collective adaptive systems such as peer-to-peer file sharing, multimedia multi-casting, aggregation in distributed databases or routing in wireless sensor networks. Despite the significance of this approach to apply collective adaptive systems in practice, applications based on overlay networks often result in a complex integration of the operational logic with topological management. This approach results in low abstraction, modularity and reconfigurability of applications that require one or more overlay networks to operate. This paper challenges this design approach by introducing the notion of overlay services that provide generic application capabilities of a broad application scope enabled by one or more overlay networks. This paper contributes the multi-level conceptual architecture of ASMA that structures and guides the realisation of overlay services by using only a few lines of high-level algorithmic expressions. Two overlay services realised according to ASMA provide a proof-of-concept for the high abstraction, modularity and reconfigurability achieved in collective adaptive systems based on overlay networks.

Key words: distributed system, overlay network, overlay service, architecture, middleware, abstraction, modularity, reconfigurability

AMS subject classifications. 68M, 68U

1. Introduction. A plethora of collective adaptive systems and applications are build by software that determines virtual communication structures on top of physical networks such as the Internet, mobile or wireless sensor networks. These structures are usually referred to as 'overlay networks' and they represent interaction patterns or linking of information that mandate the operation and optimisation of a distributed application [21, 34]. For example, IP-multicasting is not widely adopted due to economic and technical factors related with security and a high protocol complexity for network service providers [5]. On the contrary, network communication can be structured in an overlay network organised in a tree topology used for multicasting multi-media content in the application-level.

Applications based on overlay networks often result in a complex integration of the operational logic with topological management. Such applications deal with topological and organisational complexity as a way to be dynamic, adaptive and capture information changes during their runtime. For example, the organisation of a tree overlay network with certain topological properties shall improve the performance of multimedia multicasting, e.g., tree balancing and constraints in node degrees [34]. This integrating design approach results in low abstraction, modularity and reconfigurability of applications that require one or more overlay networks to operate.

The design complexity of overlay networks has also raised an argument about the impact of overlay networks on the future development of the Internet and its distributed applications. There are two main opposing views in this argument: The *purists* view overlay networks as testbeds used for the implementation and experimentation of novel Internet architectures. Purists do not view overlay networks as viable or coexisting architectural elements of the future Internet. In contrast, *pluralists* envision overlay networks as a possible solution to deal with the heterogeneity of applications and the business challenges of network service providers related to the development and adoption of technological innovations in the Internet infrastructure. This paper reasons about "*a philosophical revolution in how developers use overlays, rather than a technical alteration in how they build them*" [1]. Given that applications of collective adaptive systems based on overlay networks emerge faster than the adoption of overlay networks in the Internet infrastructure, it is evident that a higher abstraction, modularity and reconfigurability for services of overlay network is required as identified in earlier work [11, 17, 20, 18, 33, 4, 16, 41, 42].

This paper introduces the notion of an 'overlay service' that is a distributed stand-alone software system, e.g., middleware, that is based on one or more overlay networks and provides generic application capabilities

*Professorship of Computational Social Science, ETH Zurich, Zurich, Switzerland (epournaras@ethz.ch).

of a broad application scope. This paper contributes ASMA, the *Adaptive Self-organisation in a Multi-level Architecture* that is a new conceptual multi-level architecture to design and prototype overlay services [25]. A higher abstraction, modularity and reconfigurability are these novel qualitative properties that collective adaptive system inherit when designed according to ASMA. The applicability of the ASMA architecture is challenged by illustrating the modularity and reconfigurability in two architectural realisations of overlay services. These realisations concern earlier work on large-scale networked systems that perform highly complex functionality in a fully decentralised and collective fashion. However, in the new context of this paper it is shown how only a few lines of high-level algorithmic logic defined by ASMA guide and unravel the design and prototyping process of these complex adaptive systems. In addition, experimental evaluation of the overlay services provides a proof-of-concept for the high abstraction, modularity and reconfigurability achieved with the ASMA architecture. Yet, evaluation also dissects the performance trade-offs made as a result of introducing generic collective adaptive systems.

This paper is outlined as follow: Section 2 defines the main concepts proposed in this paper. Section 3 illustrates an overview of the ASMA architecture. Section 4 introduces the three levels of the ASMA architecture and their interactions. Section 5 shows how overlay services can be realised by the ASMA architecture. In the same section, the high abstraction, modularity and reconfigurability of the ASMA overlay services are studied experimentally. Section 6 compares ASMA with related work. Finally, Section 7 concludes this paper and outlines future work.

2. Overlay Networks and Services. In the context of this paper, a collective adaptive system of overlay networks consists of the following computing entities[1] as illustrated in Figure 2.1:

1. *Host*: This entity is a physical machine with a network interface connected to a physical network.
2. *Peer*: This entity is a software environment that hosts agents and enables their communication.
3. *Agent*: This entity is a software system that carries out, with some degree of independence or autonomy, a set of operations defined by a distributed application.
4. *Node*: This entity is a logical abstraction and representation of an agent in an overlay network.

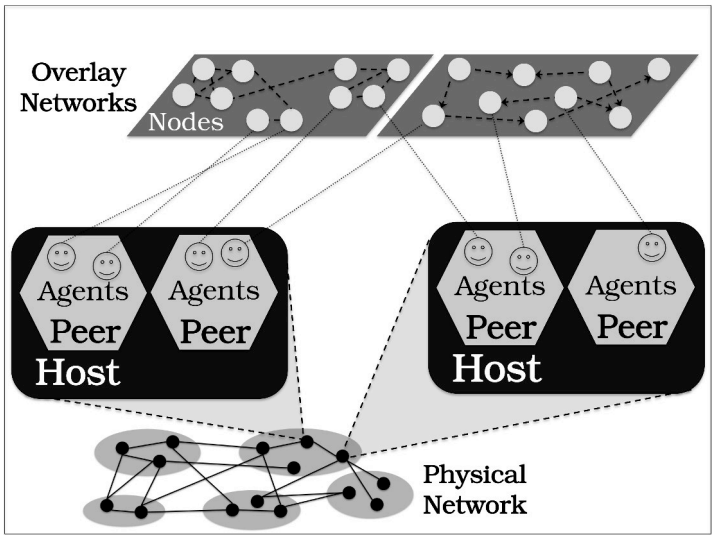


FIG. 2.1. *The four entities defined within the context of collective adaptive systems built with overlay networks: (i) host, (ii) peer, (iii) agent and (iv) node.*

An *overlay network* is defined in this paper as a graph representation of information managed by the agents of a collective adaptive system. Peers and agents can be part of a middleware system or integral parts of

[1]These overloaded terms may be used in different ways in various computing areas such as middleware, peer-to-peer, multi-agent and telecommunication systems. For example, overlay networks are built by virtual nodes that appear to be related to 'peers', 'agents', 'hosting machines' or 'software clients' in literature.

distributed applications. As shown in Figure 2.1, a host may contain more than one peer that each may also contain multiple agents. Finally, note that an overlay network is defined by agent memberships: Every agent in a network stores unique network identifiers and other information of other agents in a limited (partial) set. The memberships known to an agent are its partial *view* of the system.

This paper studies collective adaptive systems of overlay networks designed to serve a wide range of distributed applications. These generic systems are referred to in this paper as overlay services[2]. An *overlay service* is defined as a decentralised software system that provides a number of collective intelligence capabilities of a broad application scope enabled by one or more overlay networks. Overlay services can be realised as distributed middleware systems. Section 5 illustrates two representative examples of overlay services: (i) self-organisation of overlay networks in tree topologies [30] and (ii) aggregation of dynamically changing information distributed in the network [29].

An overlay service is characterised by its quality. The *quality of an overlay service* is defined as a measurable metric that quantifies the degree to which this overlay service can meet certain application objectives [19], for instance, the average response time [22] of queries to a directory service that relies on an overlay network.

3. An Architecture for Overlay Services. This paper introduces ASMA, the Adaptive Self-organisation in a Multi-level Architecture. ASMA is a conceptual self-organisation architecture with which different generic overlay services can be designed. The implementation of an individual overlay service in the ASMA architecture is referred to as *architectural realisation*. An architectural realisation entails the realisations of tasks defined in ASMA.

ASMA addresses the challenges of abstraction, modularity and reconfigurability in overlay services by introducing (i) a multi-level architecture and (ii) inter-level interactions. The complexity of a collective adaptive system is managed by multiple application-independent levels, each with a specific self-organisation goal. Each level in ASMA supports the level above and configures the level below. These bottom-up and top-down inter-level interactions tune the self-organisation operations in each level to improve the quality of an overlay service.

Figure 3.1 illustrates the ASMA architecture positioned in a single peer. The design of an overlay service in ASMA is defined by sets of (i) *criteria* and (ii) *samples* within three application-independent reconfigurable self-organisation levels: (i) the *discovery level*, (ii) the *structuring level* and (iii) the *coordination level*. A *criterion* is runtime feedback information that parametrises the operation of a level. A *sample* is continuously updated information required for the operation of a level in the ASMA architecture. For each set of *criteria* at each level in the architecture, a set of *samples* is generated. The *discovery level* discovers required information in the network. The *structuring level* structures this information. Finally the *coordination level* uses the structured information to build and provide the intended functionality to an application.

Each level of this architecture is managed by one or more autonomous agents. Two corresponding levels in two different peers are able to communicate remotely. In other words, agents of the same type are able to remotely interact. ASMA defines two types of interactions: (i) vertical and (ii) horizontal. A *vertical interaction* is the (local) exchange of *criteria* and *samples* between two different levels of ASMA located within the same peer. In contrast, a *horizontal interaction* is the (remote) exchange of *criteria* and *samples* between the same two levels of ASMA located within two different peers.

Criteria are provided in a top-down fashion in vertical interactions: from each level to the level below. In horizontal interactions, *criteria* are provided by a remote corresponding level. A *criterion* is generated based on some given *samples*. For example, *criteria* can be generated by a change in the value of a monitored metric, changes in the underlying network or the result of an agent negotiation. ASMA defines three types of *criteria* in its vertical interactions:

- *Organisational criteria*: These are *criteria* that parametrise the self-organisation operation of an overlay service. They are externally provided by an application as input to the *coordination level*.

[2]In contrast to overlay services as defined in this section, *service overlay networks* (SONs), introduced by [39], refer to a number of dedicated hosts in ISPs that explicitly allocate resources for peer-to-peer or other decentralised systems. Bandwidth resources of certain quality are provisioned based on SLAs. SONs provide a generic model for the allocation of Internet resources to decentralised systems and applications rather than a methodology of how to provide generic application capabilities enabled by overlay networks. Note that this distinction is identified in earlier work [12] that refers to overlay services as *overlay-based services*. Overlay services can coexist on top of SONs. In this case, SONs provide a business model for the Internet resource allocation required for overlay services and their applications [6].

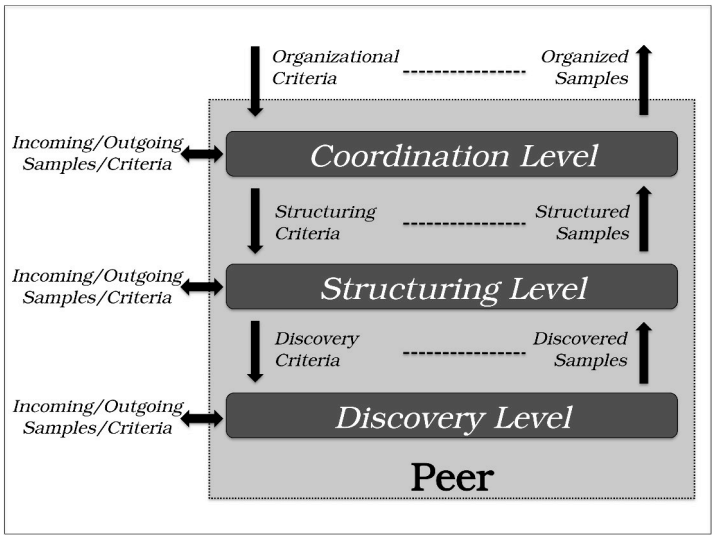


FIG. 3.1. *The three levels of the ASMA architecture in a peer.*

- *Structuring criteria*: These are *criteria* that parametrise the structuring of *samples* to improve the quality of an overlay service. They are provided by the *coordination level* to the *structuring level*.
- *Discovery criteria*: These are *criteria* that parametrise the dissemination and collection of *samples* in a network. They are provided by the *structuring level* to the *discovery level*.

Samples are provided in a bottom-up fashion in vertical interactions: from each level to the level above. In horizontal interactions, *samples* are provided by a remote corresponding level. A *sample* is defined within the context of an overlay service and may represent a wide range of information related to any of the entities of Figure 2.1: from information about the local host, such as its IP address or its geographic location, to information about the user, such as his/her reputation and trust in an online community. This information is usually abstracted from the application. For example, the reputation of a user in an online community can be represented as an abstract rank value of a node in the overlay network of this community. Three types of *samples* are defined in the vertical interactions of ASMA:

- *Discovered samples*: These are *samples* discovered in the network that are locally provided to the *structuring level* in which they are managed.
- *Structured samples*: These are *discovered samples* required for building an overlay service. They are provided by the *structuring level* to the *coordination level*.
- *Organised samples*: These are the output *samples* achieving a certain quality of an overlay service. They are provided by the *coordination level* to applications.

The *criteria* and *samples* illustrated above are application-independent and are abstracted from the application. *Samples* are provided from the one level to the other if a condition is satisfied. *Criteria* are feedback parametrisation triggered by the consumption of *samples* in a level of the architecture. The *criteria* and *samples* exchanged in horizontal interactions are referred to as (i) *incoming criteria*, (ii) *incoming samples*, (iii) *outgoing criteria* and (iv) *outgoing samples*. The semantic of these *samples* and *criteria* is defined by an architectural realisation.

In the rest of this paper, the ASMA architecture and its realisations are illustrated via high level algorithmic expressions that aim at guiding the prototyping process and structuring the different interactive levels in an overlay service and their interactions[3].

[3]The exact algorithms concerning the functionality of the overlay services are out of the scope of this paper and are illustrated in earlier work about these overlay services.

4. Architectural Levels. Each individual level of ASMA is defined according to Figure 4.1. Algorithms 1 and 2 provide an abstraction for the event generations and reactions in each ASMA level. A realisation of these algorithms is illustrated in Algorithms 3 and 4 of Section 4.1. Assume an arbitrary ASMA level that (i) generates some arbitrary output *criteria* and *samples* and (ii) reacts to some arbitrary input *criteria* and *samples*. Input *criteria* trigger execution of the `adapt` task that generates the output *samples*. Similarly, input *samples* trigger execution of the `consume` task that generates output *criteria*. The `provide` task sends the output *samples* to the level above and the `configure` task sends the output *criteria* to the level below. *Samples* and *criteria* can also be sent to a remote corresponding level in a horizontal interaction. However, the `provide` task may call the `consume` task of the same level instead of the one in the level above, suggesting in this way the possibility of internal feedback loops within each level. Similarly, the `configure` task may call the `adapt` task of the same level instead of the one in the level below. These internal calls, within a level, are possible options defined within a realisation of an ASMA level. The realisation of a level is defined by the implementation and scheduling of its tasks.

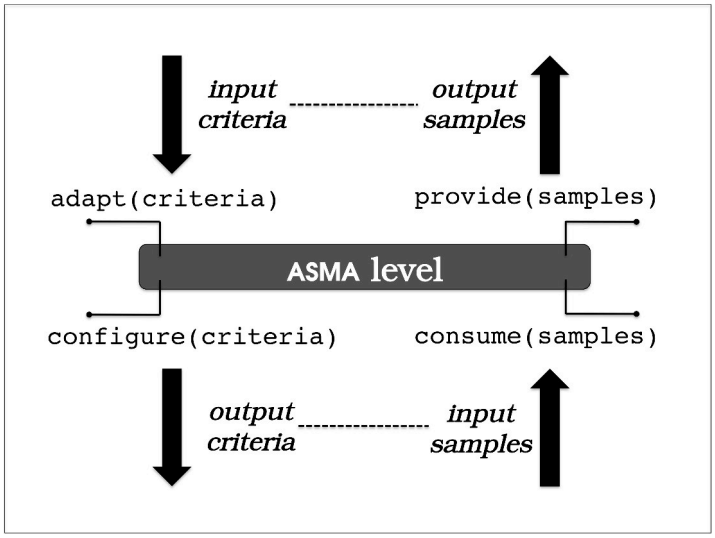


FIG. 4.1. *The executed tasks of an abstract ASMA level.*

Algorithm 1 Generations of output events in an ASMA level.

1: **while** a condition is satisfied **do**
2: `provide`(*samples*)
3: **end while**
Ensure: output

Algorithm 2 Reactions of input events in an ASMA level.

Require: input
1: **if** input=*criteria* **then**
2: *samples*=`adapt`(*criteria*)
3: **else** // input=*samples*
4: *criteria*=`consume`(*samples*)
5: `configure`(*criteria*)
6: **end if**

The three levels of ASMA are summarised as follows: The *discovery level*, positioned at the bottom of the architecture, performs discovery of remote *samples* required by an overlay service. *Sample* discovery is achieved by horizontal interactions that disseminate *outgoing criteria* and trigger the remote collection of *incoming samples* from the network. The dissemination and collection of *samples* is parametrised by the *discovery*

criteria received from the *structuring level*. The *structuring level* structures the *discovered samples* provided by the *discovery level* or other *incoming samples* received by horizontal interactions. The *structuring criteria* customise the structuring and selection of the *structured samples* provided to the *coordination level*. Finally, the *coordination level* coordinates the main functionality of an overlay service according to a set of *organisational criteria*. The *coordination level* uses the *structured samples* or other *incoming samples* received by horizontal interactions to update the *organised samples* provided to an application. The quality of an overlay service achieved with certain *structured samples* is evaluated resulting in a new set of *structuring criteria* that improve *structured samples*. The *organised samples* are a result of a continuous inter-level interactions and adaptations between the three levels of ASMA.

To certain extent, the multi-level architecture of ASMA resembles the simple architectural principle of OSI layering. However, collective adaptive systems built at the application level with overlay networks continue to integrate topological management with their main operational logic resulting in limited abstraction, modularity and reconfigurability. Therefore an OSI-like architectural principle with self-adaptive levels [43, 44] is a promising novel design approach to tackle this challenge. The modularity of the ASMA architecture could be extended to more than three levels assembled in various architectural patterns as shown in earlier work [32]. Each architectural level of ASMA is abstracted as shown in Figure 4.1, therefore, more levels could be added in the ASMA stack. This paper focuses on three levels as the overlay service realisations illustrated in Section 5 show empirically that three levels are a cost-effective compromise to model the complexity of collective adaptive systems operating in decentralised networked environments. The rest of this section illustrates each level of the ASMA architecture in detail.

4.1. The *discovery level*. The *discovery level* is responsible for the distribution and availability of *samples* to every peer of the hosts of a network providing the abstraction of *sample* discovery in the *structuring level*. The dissemination of *outgoing samples* in the network is performed using *outgoing criteria* and the `configure` task. Symmetrically, the collection of remote *incoming samples* triggers the `consume` task. Furthermore, the execution of the `adapt` task is triggered by *incoming criteria* and by the *discovery criteria* that configure the dissemination and collection of *samples* in favour of the *structuring level*. Remote communication between peers is possible as the *samples* disseminated and collected in the network contain routing information, e.g. the IP address and port number.

Making distributed *samples* locally available to the peers of a network is challenging and crucial for building decentralised overlay services. Middleware systems based on centralised information lookup or distributed lookup mechanisms designed with specific applications types in mind cannot always support scalable and generic overlay services. ASMA introduces the *discovery level* in the foundations of the architecture to bridge the information gap of decentralisation in overlay networks.

This paper focuses on a gossip-based realisation of the *discovery level*[4]. Gossiping is a simple and generic probabilistic communication model according to which agents exchange *samples* in a 'push', 'pull', or 'push-pull' fashion [14]. The exchange of *samples* is random to certain degree but other more intelligent policies can be applied as well. Gossiping information is spread in an epidemic fashion within a network [36, 35]. Furthermore, gossiping is able to prevent clustering of a network by cascading failures of its hosts.

Algorithm 3 and 4 illustrate a high-level description of a 'push-pull' gossiping protocol that realises the horizontal interactions of the *discovery level*. The *outgoing criteria* represent a 'push' message and the *incoming samples* represent a 'pull' message. Both contain local *discovered samples* exchanged in a gossiping fashion. A gossip-based *discovery level* periodically sends *outgoing criteria* to a remote *discovery level* of a selected peer defined within these *criteria* (line 3 of Algorithm 3). The *discovered samples* are also provided periodically to the *structuring level* (line 2 of Algorithm 3). Furthermore, the *discovery level* reacts to *incoming criteria* by adapting its *discovered samples* and providing in return *outgoing samples* (line 2-3 of Algorithm 4). The *incoming samples* are consumed and generate the next *outgoing criteria* (line 5 of Algorithm 4).

The core gossiping operations are performed in the `adapt` and `consume` tasks illustrated in Algorithm 5 and 6. The `adapt` task (i) handles *incoming criteria* that are actual 'push' gossiping messages and (ii) generates

[4]Gossiping is chosen because of its robustness properties and the rapid dissemination of information in distributed networks. These properties benefit the overlay services studied in this paper. Other mechanisms can be employed as well, e.g., flooding [15], random walks [8] and DHT overlays [38].

Algorithm 3 Event generations by a gossip-based *discovery level*.

```
1: loop // periodically
2:     provide(discovered samples)
3:     configure(outgoing criteria)
4: end loop
Ensure: output
```

Algorithm 4 Event reactions by a gossip-based *discovery level*.

```
Require: input
1: if input=incoming criteria then
2:     outgoing samples=adapt(incoming criteria)
3:     provide(outgoing samples)
4: else // input=incoming samples
5:     outgoing criteria=consume(incoming samples)
6: end if
```

outgoing samples that are actual 'pull' gossiping messages (line 1-5 of Algorithm 5). The *discovery criteria* parametrise gossiping by, for example, selecting policies [14] that tune the dissemination of *samples* under various network conditions, e.g. failures in hosts. The `consume` task updates the *discovered samples* with *incoming samples* (line 1 of Algorithm 6) and selects *outgoing samples* to disseminate from the *discovered samples* (line 2 of Algorithm 6).

Algorithm 5 The `adapt` task in a gossip-based *discovery level*.

```
Require: criteria
1: if criteria=incoming criteria then
2:     outgoing samples=selectToDisseminate(discovered samples)
3:     incoming samples=getSamples(incoming criteria)
4:     outgoing criteria=consume(incoming samples)
5:     return outgoing samples
6: else // criteria=discovery criteria
7:     // Parametrises gossiping [14]:
8:     // 'peer selection', 'view propagation' and 'view selection' policies
9:     return discovered samples
10: end if
Ensure: samples
```

Algorithm 6 The `consume` task in a gossip-based *discovery level*.

```
Require: incoming samples
1: discovered samples=selectToCollect(incoming samples)
2: outgoing samples=selectToDisseminate(discovered samples)
3: outgoing criteria=getCriteria(outgoing samples)
4: return outgoing criteria
Ensure: outgoing criteria
```

Finally, the `selectToCollect` and `selectToDisseminate` tasks (line 1 and 2 of Algorithm 6) implement the push-pull gossiping interactions and correspond to the respective selection tasks of the peer sampling service [14]. The `getCriteria` task performs the selection of the agent to gossip with (line 3 of Algorithm 6) and the `getSamples` task simply derives the *incoming samples* included in a set of *incoming criteria* (line 3 of Algorithm 5).

4.2. The *structuring level*. The *structuring level* is responsible for the management of *discovered samples*, providing in this way an abstraction to the *coordination level*. More specifically, the *structuring level* performs (i) structuring, such as sorting, clustering and classification of the *discovered samples* received from the *discovery level* and (ii) selection of the *structured samples* provided to the *coordination level*.

Structuring and selection are based on criteria defined by an adaptation strategy. The following three examples illustrate some adaptation strategies:

- Sorting a list of ranked *samples* in an ascending order and selecting the first *sample* from the list.
- Clustering a set of ranked *samples* based on their ranking distance and selecting the highest or lowest ranked *sample* in each cluster.

- Classifying a set of *samples* in a number of classes and selecting the most recently added *sample* from each class.

There is a wide range of adaptation strategies that can be designed regarding a certain self-organisation goal of an overlay service. Adaptation strategies provide dynamic management of *samples* as:

- Multiple adaptation strategies can be adopted dynamically during runtime.
- The parameters of an adaptation strategy that define the structuring and selection of *samples* can be reconfigured during runtime.

Both approaches are based on feedback received within the *structuring criteria* from the *coordination level*. The *structuring criteria* result in new *structured samples* that potentially improve the quality of an overlay service. In this case, the *structuring criteria* are an actual feedback about the provided *structured samples* that can be used by the *structuring level* for either (i) switching to a different adaptation strategy or (ii) reconfiguring the parameters that define a certain adaptation strategy.

Algorithms 7 and 8 illustrate the event generations and reactions in the *structuring level*. The tasks executed by the *structuring level* are specialisations of an abstract ASMA level. Optionally, *criteria* and *samples* can be exchanged via horizontal interactions.

Algorithm 7 Event generations by the *structuring level*.

```
1: while a condition is satisfied do
2:     provide(structured samples)
3:     provide(outgoing samples) // Optional, defined in a level realisation
4:     configure(outgoing criteria) // Optional, defined in a level realisation
5: end while
Ensure: output
```

Algorithm 8 Event reactions by the *structuring level*.

```
Require: input
1: if input=structuring criteria then
2:     structured samples=adapt(structuring criteria)
3: else if input=discovered samples then
4:     discovery criteria=consume(discovered samples)
5:     configure(discovery criteria)
6: else // input=incoming criteria or incoming samples
7:     // Optional, defined in a level realisation
8: end if
```

Algorithms 9 and 10 illustrate the `adapt` and `consume` task in the *structuring level*. The `adapt` task is based on two subtasks, the `adopt` and `selectToProvide`. The first subtask is responsible for the choice and reconfiguration of the adaptation strategy based on which the *structured samples* are selected (line 2 of Algorithm 9). A learning or rule-based system may be used to correlate certain feedback information contained in the *structuring criteria* with a number of adaptation strategies supported by the *structuring level*. Furthermore, the `selectToProvide` subtask selects a number of *structured samples* provided to the *coordination level* (line 3 of Algorithm 9). The selection of *structured samples* is performed based on criteria defined within the adopted adaptation strategy. The `consume` task defines the structuring of the *discovered samples*, such as sorting, clustering, classification, etc., based on the adopted adaptation strategy (line 2 of Algorithm 10). The `adopt`, `selectToProvide` and `structure` subtasks are realised by each overlay service and therefore their definition is subject of an architectural realisation

Algorithm 9 The `adapt` task in the *structuring level*.

```
Require: criteria
1: if criteria=structuring criteria then
2:     strategy=adopt(structuring criteria)
3:     structured samples=selectToProvide(strategy)
4:     return structured samples
5: else if criteria=incoming criteria then
6:     // Optional, defined in a level realisation
7: end if
Ensure: structured samples
```

Algorithm 10 The `consume` task in the *structuring level*.

Require: *samples*
1: **if** *samples*=*discovered samples* **then**
2: *discovery criteria*=`structure`(strategy, *discovered samples*)
3: **return** *discovery criteria*
4: **else if** *samples*=*incoming samples* **then**
5: // Optional, defined in a level realisation
6: **end if**
Ensure: *discovery criteria*

Adaptation strategies introduce a modularity level in the design phase of an overlay service. By adopting multiple adaptation strategies or by reconfiguring a certain strategy, self-organisation provides a flexible compositional environment for meeting complex organisational goals related to various criteria of complex adaptive systems.

4.3. The *coordination level*. The *coordination level* is responsible for the continuous organisational update of the *organised samples* provided to an application. The update of the *organised samples* is based on the *structured samples* provided by the *structuring level* and the *organisational criteria* provided by an application. Updating the *organised samples* may require some coordination between remote agents of the *coordination level*. These agents are defined within the *structured samples*. Coordination may concern the exchange of *samples* required for the operation of an overlay service, a negotiation between two agents about their required *samples*, a query, or some other type of remote interaction and operation.

The *organised samples* can be tuned by a fitness function [23] or another evaluation scheme that maximises the quality of an overlay service. This process generates a set of *structuring criteria* containing feedback for the *structuring level* to trigger the next *structured samples* that improve the quality of an overlay service. Therefore, the exchange of *samples* and *criteria* between the *structuring level* and the *coordination level* is a continuous and iterative optimisation process of the quality of an overlay service.

Algorithm 11 illustrates the event generations in the *coordination level*. The delivery of the *organised samples* to the application is governed by the *organisational criteria* (line 1 and 2 in Algorithm 11). The *organisational criteria* related with this delivery may concern a certain quality of an overlay service, or an elapsed runtime period.

Algorithm 11 Event generations by the *coordination level*.

1: **while** a condition is satisfied **do**
2: `provide`(*organised samples*)
3: **end while**
Ensure: *organised samples*

Algorithm 12 shows the event reactions in the *coordination level*. A simple coordination scenario of horizontal interactions is assumed in which an agent of the *coordination level* sends a set of *outgoing criteria* containing some *outgoing samples* and receives back *incoming samples*. This scenario corresponds to coordination based on an information exchange. Consuming *structured samples* triggers the *outgoing criteria* (line 2 in Algorithm 12) and a set of *incoming criteria* results in providing *outgoing samples* to the agent from which these *criteria* are received (line 8 and 9 in Algorithm 12). Receiving *incoming samples* completes the coordination by sending a set of *structuring criteria* to the *structuring level* (line 5 and 6 in Algorithm 12). Finally, the *organisational criteria* adapt the *organised samples* (line 11 in Algorithm 12) by parametrising the operation of an overlay service.

Algorithm 13 illustrates the `adapt` task. The `adapt` task performs the parametrisation of the *coordination level* as defined in the *organisational criteria* (line 2 in Algorithm 13). It also handles coordination by (i) consuming the *incoming samples* contained in a set of received *incoming criteria*, (ii) configuring the *structuring level* with the *structuring criteria* and (iii) generating the *outgoing samples* that is sent back to the agent that initiates coordination (line 4-9 in Algorithm 13).

The `consume` task, illustrated in Algorithm 14, (i) initiates the coordination by generating *outgoing criteria* (line 2 in Algorithm 14) and (ii) organises the *organised samples* provided to the application (line 5 in

Algorithm 12 Event reactions by the *coordination level*.

```
Require: input
 1: if input=structured samples then
 2:     outgoing criteria=consume(structured samples)
 3:     configure(outgoing criteria)
 4: else if input=incoming samples then
 5:     structuring criteria=consume(incoming samples)
 6:     configure(structuring criteria)
 7: else if input=incoming criteria then
 8:     outgoing samples=adapt(incoming criteria)
 9:     provide(outgoing samples)
10: else // input=organisational criteria
11:     organised samples=adapt(organisational criteria)
12: end if
```

Algorithm 13 The `adapt` task in the *coordination level*.

```
Require: criteria
 1: if criteria=organisational criteria then
 2:     organised samples=parameterise(organisational criteria)
 3:     return organised samples
 4: else if criteria=incoming criteria then
 5:     incoming samples=getSamples(incoming criteria)
 6:     structuring criteria=consume(incoming samples)
 7:     configure(structuring criteria)
 8:     outgoing samples=finaliseCoordination(structuring criteria)
 9:     return outgoing samples
10: end if
Ensure: samples
```

Algorithm 14). This task results in the *structuring criteria* provided to the *structuring level* as feedback for the improvement of the quality of an overlay service.

Algorithm 14 The `consume` task in the *coordination level*.

```
Require: samples
 1: if samples=structured samples then
 2:     outgoing criteria=initialiseCoordination(structured samples)
 3:     return outgoing criteria
 4: else // samples=incoming samples
 5:     structuring criteria=organise(incoming samples)
 6:     return structuring criteria
 7: end if
Ensure: criteria
```

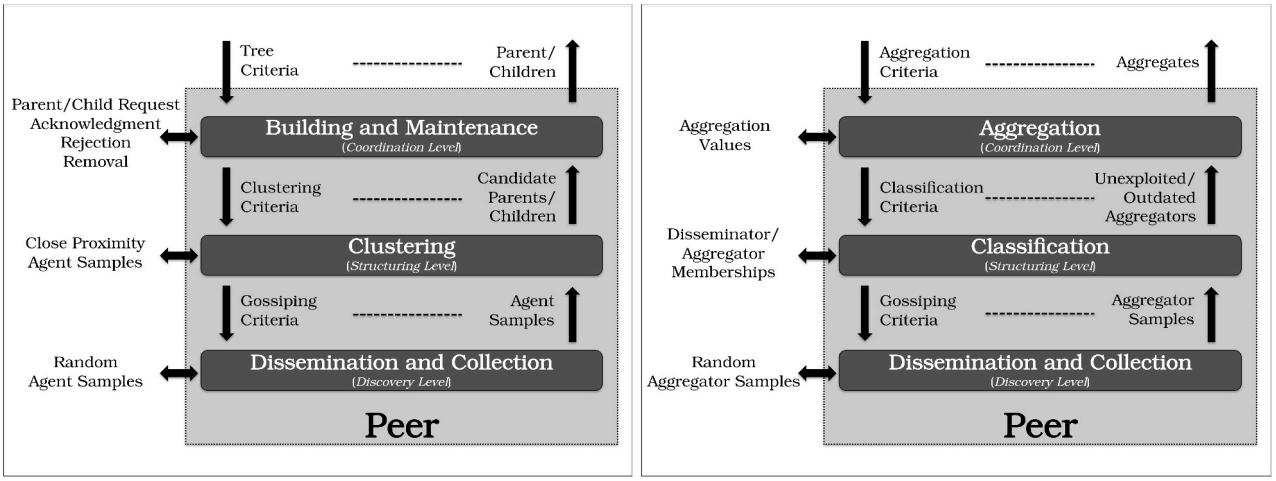
The `adapt` and `consume` tasks show that coordination is an actual response of *outgoing samples* to a set of *incoming criteria* based on the *structuring criteria* (line 8 in Algorithm 13) and *structured samples* (line 2 in Algorithm 14) respectively. Note that the `initialiseCoordination`, `finaliseCoordination`, `organise` and `parameterise` subtasks are realised within an overlay service.

5. Architectural Realisations. This section illustrates two overlay services designed and realised according to the ASMA architecture:

- AETOS, the Adaptive Epidemic Tree Overlay Service [30].
- DIAS, the Dynamic Intelligent Aggregation Service [29].

These two overlay services have a generic application scope, yet, earlier work illustrates evidence about their applicability in the domain of demand-side energy management [25, 28, 31]. Figure 5.1 illustrates the realized architectures of these overlay services. Both architectures follow the design paradigm of ASMA illustrated in Figure 3.1.

AETOS self-organises overlay networks in various tree topologies to meet different application requirements. Trees are used for operations such as decision-making, aggregation, information dissemination etc., with an applicability in a wide range of distributed applications, e.g., distributed databases [40, 9] and multimedia multicasting [34]. The three levels of ASMA are relevant to model the complexity of such an overlay service: (i) discovery for accessing every possible peer in the network, (ii) structuring and selecting candidate parents/children according to the intended built topology and (iii) coordination between the parents/children



(a) The AETOS overlay service

(b) The DIAS overlay service.

FIG. 5.1. *Two overlay service realisations of the ASMA architecture.*

to form bidirectional links. Building and maintenance of a tree topology is entirely performed by the AETOS overlay service without involvement of applications that use it.

DIAS computes aggregates, such as AVERAGE, SUMMATION, MAXIMUM, and STANDARD DEVIATION, of dynamically changing values distributed in every peer of an unstructured overlay network. Collective computation of aggregates benefits a wide range of distributed applications as aggregates are used for load-balancing [26], data mining [37], sensor networks [3] and other. Accurate computations of aggregates are achieved as values that are duplicate or locally changing are detected within the three-level architecture of ASMA: (i) discovery of information from every possible peer, (ii) structuring this information as exploited (duplicate information), unexploited (new information) or outdated (changed information) and (iii) using this information for coordinating accurate computations of aggregates between peers. Periodic computations of system-wide aggregate information are entirely performed by the DIAS overlay service without involvement of applications that use it.

This paper shows how the collective adaptive processes of each of AETOS and DIAS are dissected and realised within the defined tasks of the ASMA architecture. In other words, these overlay services are used as case studies of ASMA. Although the main functionality of AETOS and DIAS is illustrated in earlier work, this paper contributes the novel design approach of ASMA that justifies the higher modularity and reconfigurability in both of these overlay services. This section recalls experimental results of AETOS and DIAS to show how the design methodology of ASMA empirically justifies the high modularity and reconfigurability of these overlay services. Therefore, the earlier experimental results provide new findings in the new context of this paper. These results validate the reconfigurability of overlay services realised according to ASMA architecture and they also provide evidence of how the modularity concept of ASMA is applied in practice. For example, it is shown how results about the performance trade-offs observed in AETOS and DIAS are justified by the reconfigurability engineered in ASMA. Similarly, the performance results of the adaptation strategies provide evidence that overlay services can provide a broad range of modular application capabilities.

The overlay services are implemented and evaluated in the Protopeer prototyping toolkit according to the ASMA architecture [7]. Protopeer is set up to simulate networks of 1500 peers. Each peer hosts three agents[5] that implement the architectural levels of ASMA. Each experiment has a duration calculated in *epochs*. Each epoch in Protopeer lasts 1000 ms.

5.1. AETOS: The Adaptive Epidemic Tree Overlay Service. The *discovery level* of AETOS is realised by the gossiping protocol of the peer sampling service [14] as illustrated in Section 4.1. Gossiping guarantees that a tree topology can reconnect after single node failures as agents continuously exchange information

[5]The *peerlets* of Protopeer implement the agents of each ASMA level.

to discover each other. The nodes of AETOS are ranked with a weight to order their positioning in the formed tree. The weight of the nodes, together with the IP address and the port number, are part of the *discovered samples*.

The *structuring level* clusters *discovered samples* received from the *discovery level* into a *proximity view* $\mathbf{v}_i(proximity)$ that is a list of candidate parents and children. Eight adaptation strategies [27] define criteria for clustering based on the proximity of the node, e.g., minimum or maximum euclidean distance of their weights. In addition, adaptation strategies periodically select and provide the candidate parent and children with the closest proximity to the *coordination level*. Their quality is evaluated by the *coordination level* and feedback is provided to the *structuring level* via *organisational criteria*. Event generations of the *structuring level* are time-based as shown in line 1 of Algorithm 7.

Clustering is tuned by employing the T-MAN mechanism [13], that introduces horizontal interactions between the agents of the *structuring level*. The main intuition behind the introduction of T-MAN is to improve the quality of clustering by letting close proximity agents exchange their *discovered samples* in which they share interest. Algorithms 15 and 16 illustrate the T-MAN functionality embedded in the `adapt` and `consume` task of the *structuring level*. The functionality of T-MAN is injected in the optional blocks of Algorithm 9 and 10. Furthermore, the event generations of T-MAN are periodic, meaning that the `configure`(*outgoing criteria*) at line 4 of Algorithm 7 is executed periodically.

Algorithm 15 T-MAN functionality embedded in the `adapt` task of the *structuring level* in AETOS.

Require: *criteria*
1: **if** *criteria*=*incoming criteria* **then**
2: *incoming samples*=`getSamples`(*incoming criteria*)
3: *discovery criteria*=`structure`(strategy, *incoming samples*)
4: `configure`(*discovery criteria*)
5: *outgoing samples*=`selectToCluster`($\mathbf{v}_i(proximity)$)
6: **return** *outgoing samples*
7: **end if**
Ensure: *outgoing samples*

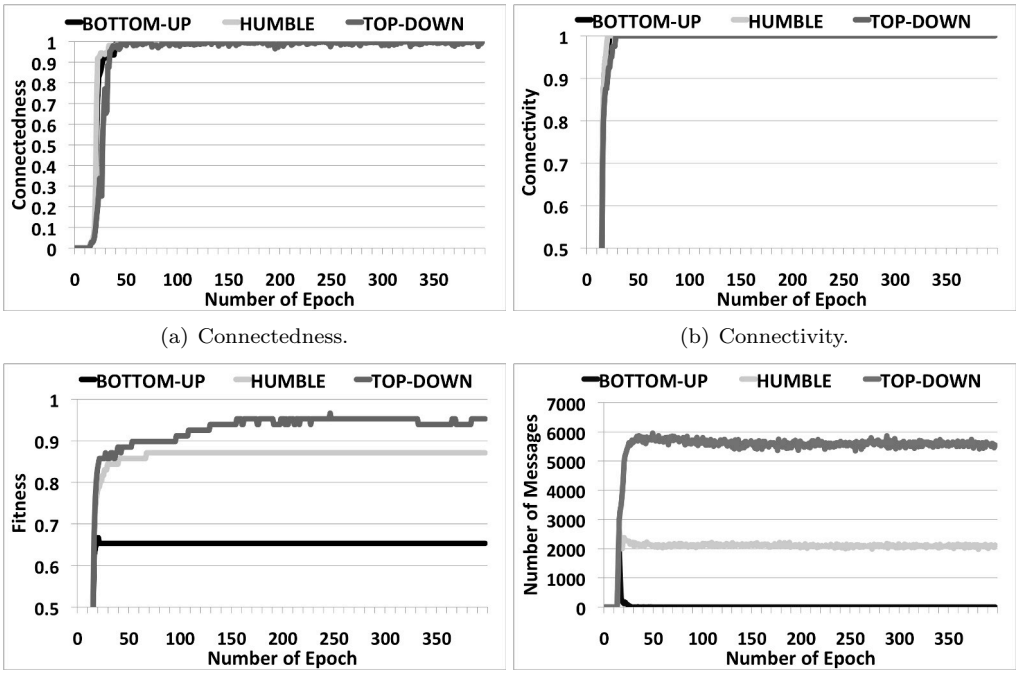
Outgoing criteria and *outgoing samples* contain the exchanged proximity views. The `selectToCluster` subtask selects the ranked node with which the proximity views are exchanged. It also embeds the proximity view in the *outgoing samples* (line 5 and 3 of Algorithm 15 and 16). The `structure` subtask (line 3 and 2 of Algorithm 15 and 16) fills the proximity view according to the adopted adaptation strategy [27].

Algorithm 16 T-MAN functionality embedded in the `consume` task of the *structuring level* in AETOS.

Require: *samples*
1: **if** *samples*=*incoming samples* **then**
2: *discovery criteria*=`structure`(strategy, *incoming samples*)
3: *outgoing samples*=`selectToCluster`($\mathbf{v}_i(proximity)$)
4: *outgoing criteria*=`getCriteria`(*outgoing samples*)
5: **return** *discovery criteria*
6: **end if**
Ensure: *discovery criteria*

The *coordination level* is responsible for building and maintaining the tree topology. Agents negotiate the formation of bidirectional links with the candidate parents and children received from the *structuring level*. Four types of messages are exchanged that realise horizontal interactions between the agents of the *coordination level*: *request*, *acknowledgment*, *rejection* and *removal*. The *request*/*removal* messages are used as *outgoing criteria* and the *acknowledgment*/*rejection* messages as *outgoing samples*. In both cases, the weights of the communicating agents are included using the `getCriteria` and `getSamples` subtasks. The establishment of a link results in *structuring criteria* with a positive feedback to the *structuring level*. In contrast, the rejection of a *request* or the *removal* of a link results in negative feedback. Based on this feedback, the clustering performed in the *structuring level* is tuned to provide candidate parents and children that improve the performance of AETOS [30]. The protocol interactions are realised within the `initialiseCoordination` and `finalise-Coordination` subtasks of Algorithm 14 and 13. Finally, the `organise` subtask of Algorithm 14 facilitates the coordination logic by reacting to the received messages.

Figure 5.2 illustrates performance trade-offs between three adaptation strategies of AETOS[6]: BOTTOM-UP, HUMBLE and TOP-DOWN. Four performance metrics are shown that measure the quality of the AETOS overlay service: (i) *connectedness*, (ii) *connectivity*, (iii) *fitness* and (iv) *communication cost*. Connectedness measures how well connected nodes are in a single tree, whereas, connectivity measures the extent to which nodes establish the maximum number of links (children) that their resources allow. Fitness evaluates the sorting of the nodes in a tree and the communication cost counts the number of messages exchanged at the *coordination level*.



(a) Connectedness.

(b) Connectivity.

(c) Fitness.

(d) Communication cost.

FIG. 5.2. *Performance trade-offs for three adaptation strategies of AETOS:* BOTTOM-UP, HUMBLE *and* TOP-DOWN *[30].*

Figure 5.2a and 5.2b show that all three strategies build a well-connected tree topology within a few epochs. However, TOP-DOWN has the highest fitness of over 0.95 as shown in Figure 5.2c. HUMBLE and BOTTOM-UP follow with a fitness of 0.87 and 0.65 respectively. The high fitness of TOP-DOWN comes at a high communication cost of over 5000 messages per epoch as shown in Figure 5.2d. HUMBLE and BOTTOM-UP converge to 2000 and 0 messages per epoch respectively.

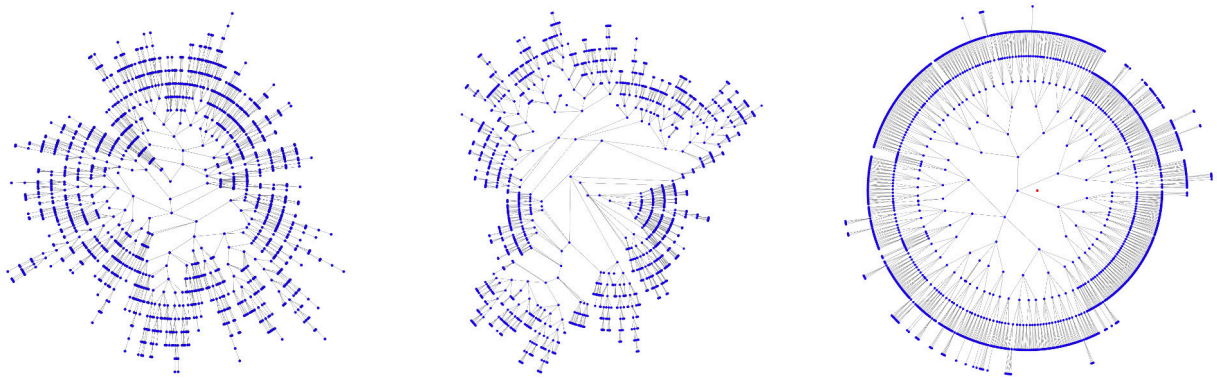
Figure 5.3 illustrates snapshots[7] of the tree topologies built by the adaptation strategies on the 350th epoch. The snapshots show visually the effect of high fitness in the tree topology built by TOP-DOWN.

These experimental results show that the ASMA architecture provides a high modularity and reconfigurability in the overlay service of AETOS. The performance of each architectural level can be separately studied and tuned, e.g., communication cost of the *coordination level* as shown in Figure 5.2d. A number of adaptation strategies provide different performance trade-offs. For example, in a network with limited bandwidth resources, BOTTOM-UP is the most effective, whereas, if network resources are not a constraint, a significantly higher fitness is achievable by TOP-DOWN.

The vertical interactions defined within the ASMA architecture can be used to manage and control these

[6]The adaptation strategies define the candidate parent and/or children with which each agent chooses to connect. BOTTOM-UP: the lowest ranked candidate parent; HUMBLE: the lowest ranked candidate parent and children; TOP-DOWN: the highest ranked candidate children.

[7]Visualisations are performed with the JUNG library [24] available at: `http://jung.sourceforge.net` (last accessed: December 2014)



(a) BOTTOM-UP. (b) HUMBLE. (c) TOP-DOWN.

FIG. 5.3. *Visualisation of three adaptation strategies of AETOS on the 350th epoch [30].*

performance trade-offs. This section illustrates a scenario in which two adaptation strategies[8], PRESBYOPIC and BOTTOM-UP, are combined in such a way that a new strategy, HYBRID-03, inherits the high performance of each individual one without their limitations. The adaptation strategies are combined via a dynamic adoption scheme in which agents of the *structuring level* change from PRESBYOPIC to BOTTOM-UP during runtime. Switching can be performed automatically by monitoring the convergence of one of the performance metrics at the *coordination level*. If the monitored metric has converged, meaning its deviations are significantly lower than the ones during system startup, a switching signal can be included in the *structuring criteria*. This section shows the feasibility of designing hybrid strategies by empirically defining the switching time point on the 250th epoch and measuring the change of fitness and communication cost. Figure 5.4 illustrates the performance of HYBRID-03 under this scenario.

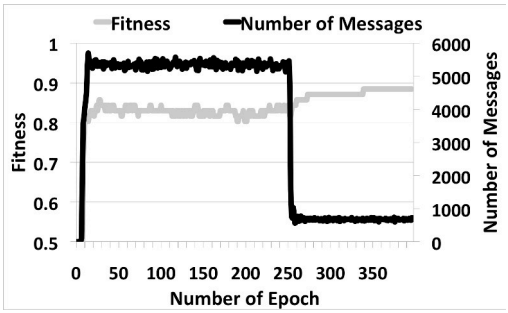
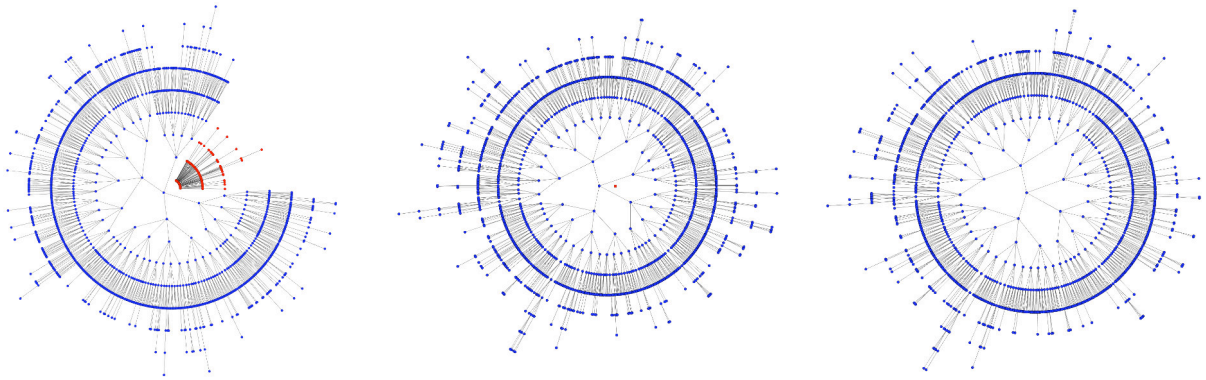


FIG. 5.4. *Perfromance of AETOS before and after switching from* PRESBYOPIC *to* BOTTOM-UP *on the 250th epoch [30].*

After switching from PRESBYOPIC to BOTTOM-UP, fitness increases 8% whereas, the number of messages drop from 5300 to 700 messages per epoch. Figure 5.5 illustrates the visualisation of the tree topology before and after switching strategies. Connectedness and connectivity are maximised. A higher fitness results in a higher number of nodes connected closer to the root of the tree after adopting BOTTOM-UP.

These performance enhancements are achieved without changing any architectural element of AETOS. Each layer realisation of ASMA retains its objective in each of the scenarios shown. ASMA provides a flexible, structured and modular architectural concept to realise and make manageable the complex system behaviour of AETOS.

[8]The agents choose to connect with the candidate parent and/or children as follows: PRESBYOPIC: the highest ranked candidate parent and the lowest ranked candidate children; BOTTOM-UP: the lowest ranked candidate parent.



(a) 245th epoch. (b) 255th epoch. (c) 380th epoch.

FIG. 5.5. *Visualisation of* HYBRID-03 *before and after switching from* PRESBYOPIC *to* BOTTOM-UP *on the 250th epoch [30].*

5.2. DIAS: The Dynamic Intelligent Aggregation Service. DIAS performs decentralised aggregation by computing aggregation functions that receive as input states distributed in the peers of a network. A *state* represents a (aggregation) value of an application parameter at a specific point in time. A node may contain the *selected state* s'_i that is the one aggregated by nodes. During system runtime, the selected state s'_i may change and is equal to one and only one state from a finite number v of locally unique *possible states* $s'_i = s^0_i|s^1_i|...|s^{v-1}_i$. As the selected state changes, an earlier selected state is indicated as $\hat{s}_i$. Each node i contains an *aggregator* A_i, a *disseminator* D_i or both. An *aggregator* computes aggregation functions as $f(s'_0, s'_1, ..., s'_{n-1})$, whereas a *disseminator* provides the selected state to *aggregators*.

The *discovery level* of DIAS is realised by the gossiping protocol of the peer sampling service [14] as illustrated in Section 4.1. Gossiping guarantees that information is disseminated to all peers in a network to achieve accurate computations of aggregation functions. Agents disseminate and collect the IP address and port number of *aggregators* to which *disseminators* send their selected state.

The *structuring level* classifies *aggregator samples* received from the *discovery level* into three classes: (i) exploited, (ii) unexploited and (iii) outdated. The exploited *aggregators* of a *disseminator* D_i are the ones that have aggregated its earliest selected state s'_i. The unexploited *aggregators* of a *disseminator* D_i are the ones with which aggregation has not been performed and therefore, D_i has not provided any of its selected states to these *aggregators*. Finally, the outdated *aggregators* of a *disseminator* D_i are the ones that have aggregated a selected state of this *disseminator* earlier but since then the selected state has changed. Two adaptation strategies, EXPLOITATION and UPDATE, provide to the *coordination level* with priority either unexploited or outdated *aggregators* respectively. A third adaptation strategy, RANDOM, performs a random selection between unexploited and outdated.

Classification in these three classes is possible via local storage of *aggregation memberships* that is a representation of the information required to perform accurate aggregation. Accuracy indicates the quality of the DIAS overlay service and concerns the computation of aggregation functions without counting twice states or counting outdated states. An aggregation membership $M_{group}(member)$ of a certain 'member' to a certain 'group' is either positive or negative. Each agent of the *structuring level* in a peer i stores unique identifiers of possible states $\mathsf{S}^0_i, ..., \mathsf{S}^{v-1}_i$ corresponding to the actual possible states $s^0_i, ..., s^{v-1}_i$. Respectively, S'_i and $\hat{\mathsf{S}}_i$ refer to the unique identifiers of the selected s'_i and outdated $\hat{s}_i$ state in peer i. The *structuring level* stores a representation of the local states, their unique identifiers, and the *coordination level* stores the actual states, e.g., numerical or other type. The *structuring level* also uses the local unique peer identifier to map the local *aggregator* A_i and *disseminator* D_i. Therefore, $\mathsf{A}_i = \mathsf{D}_i$. An *aggregator* A_j and a *disseminator* D_i in two peers i and j perform horizontal interactions to exchange the required state information for aggregation. Four local aggregation memberships are involved in the aggregation performed:

MEMBERSHIP 1 ($M_{\mathsf{D}_i}(\mathsf{A}_j)$). *Membership of an aggregator in a disseminator.*

A *disseminator* D_i stores the identifier of an *aggregator* A_j to which it has disseminated its selected state

at least once during an aggregation.

MEMBERSHIP 2 ($M_{\mathsf{S}_i^u}(\mathsf{A}_j)$). *Membership of an aggregator in a possible state.*

A *disseminator* D_i stores the identifier of an *aggregator* A_j for each possible state identified as S_i^u aggregated by this *aggregator*.

MEMBERSHIP 3 ($M_{\mathsf{A}_j}(\mathsf{D}_i)$). *Membership of a disseminator in an aggregator.*

An *aggregator* A_j stores the identifier of a *disseminator* D_i from which it has aggregated its selected state at least once during an aggregation.

MEMBERSHIP 4 ($M_{\mathsf{A}_j}(\mathsf{S}'_i)$). *Membership of a selected state in an aggregate.*

An *aggregator* A_j stores the identifier of a selected state S'_i aggregated from a *disseminator* D_i.

The *structuring level* efficiently stores these aggregation memberships in bloom filters. A *bloom filter* is a probabilistic data structure that efficiently stores membership information at a cost of false positives. The *structuring level* is able to detect and prevent inconsistencies in the aggregation originated from false positives by using mutual aggregation memberships between an *aggregator* and a *disseminator*, e.g., $M_{\mathsf{S}_i^u}(\mathsf{A}_j)$-$M_{\mathsf{A}_j}(\mathsf{D}_i)$ and $M_{\mathsf{S}_i^u}(\mathsf{A}_j)$-$M_{\mathsf{A}_j}(\mathsf{S}'_i)$ are mutual as they are representation of the same information.

Algorithm 17 and 18 illustrate the interactions of an *aggregator* A_i with a *disseminator* D_j. These interactions refer to the optional parts of the `adapt` and `consume` tasks in Algorithm 9 and 10 respectively. Aggregation is initiated by the `selectToProvide` subtask. Before a selected *aggregator* A_i is provided to the *coordination level*, a set of *outgoing criteria* is sent to A_i to make its aggregation memberships consistent to the memberships of *disseminator* D_j. Lines 1-7 of Algorithm 17 illustrate the update of the aggregation memberships by *aggregator* A_i. Update of the memberships is performed according the classification outcome of A_i by *disseminator* D_j. *Aggregator* A_i adds the memberships $M_{\mathsf{A}_i}(\mathsf{D}_j)$ and $M_{\mathsf{A}_i}(\mathsf{S}'_j)$ if it is classified as unexploited by D_j and additionally removes membership $M_{\mathsf{A}_i}(\hat{\mathsf{S}}_j)$ if it is classified as outdated. Aggregation is performed unidirectionally (flag='uni'), however, a bidirectional aggregation is performed if there is an *aggregator* A_j and a *disseminator* D_i (flag='uni-bi'). This option is checked in lines 9-18 of Algorithm 17.

Algorithm 17 Aggregation operations embedded in the `adapt` task of the *structuring level* in DIAS.

Require: *incoming criteria*: flag, class, D_j, S'_j, $\hat{\mathsf{S}}_j$
1: add $M_{\mathsf{A}_i}(\mathsf{S}'_j)$
2: **if** class=unexploited **then**
3: add $M_{\mathsf{A}_i}(\mathsf{D}_j)$
4: **end if**
5: **if** class=outdated **then**
6: remove $M_{\mathsf{A}_i}(\hat{\mathsf{S}}_j)$
7: **end if**
8: **if** flag='uni' **then**
9: **if** $M_{\mathsf{D}_i}(\mathsf{A}_j)$: *negative* **then**
10: *outgoing criteria*=`getCriteria`('bi', unexploited, D_i, S'_i, $\hat{\mathsf{S}}_i$)
11: *outgoing samples*=`getSamples`('uni-bi', class, A_i, *outgoing criteria*)
12: **else if** $M_{\mathsf{D}_i}(\mathsf{A}_j)$: *positive* **and** $M_{\mathsf{S}'_i}(\mathsf{A}_j)$: *negative* **then**
13: *outgoing criteria*=`getCriteria`('bi', outdated, D_i, S'_i, $\hat{\mathsf{S}}_i$)
14: *outgoing samples*=`getSamples`('uni-bi', class, A_i, *outgoing criteria*)
15: **else** // $M_{\mathsf{D}_i}(\mathsf{A}_j)$: *positive* **and** $M_{\mathsf{S}'_i}(\mathsf{A}_j)$: *positive*
16: *outgoing samples*=`getSamples`('uni', class, A_i)
17: **end if**
18: **return** *outgoing samples*
19: **else** // flag='bi'
20: *outgoing samples*=`getSamples`('bi', class, A_i)
21: **return** *outgoing samples*
22: **end if**
Ensure: *outgoing samples*

Similarly, the receipt of *incoming samples* by *disseminator* D_i triggers the update of its aggregation memberships as shown in lines 1-7 of Algorithm 18. *Disseminator* D_i adds the memberships $M_{\mathsf{D}_i}(\mathsf{A}_j)$ and $M_{\mathsf{S}'_i}(\mathsf{A}_j)$ if *aggregator* A_j is unexploited and additionally removes the membership $M_{\hat{\mathsf{S}}_i}(\mathsf{A}_j)$ if A_j is outdated. This completes a unidirectional aggregation. If a bidirectional aggregation is performed, the `adapt` task is executed with the *incoming criteria* as an input (line 9 of Algorithm 18).

The *coordination level* is responsible for the computation of aggregates. An aggregate is continuously computed based on an aggregation function provided by the aggregation *criteria*. The `parameterise` subtask of

Algorithm 18 Aggregation operations embedded in the `consume` task of the *structuring level* in DIAS.

Require: *incoming samples*: flag, class, A_j, *incoming criteria*
1: add $M_{S'_i}(\mathsf{A}_j)$
2: **if** class=unexploited **then**
3: add $M_{\mathsf{D}_i}(\mathsf{A}_j)$
4: **end if**
5: **if** class=outdated **then**
6: remove $M_{\hat{\mathsf{S}}_i}(\mathsf{A}_j)$
7: **end if**
8: **if** flag='uni-bi' **then**
9: `adapt`(*incoming criteria*)
10: **end if**

Algorithm 13 provides the aggregation function. Aggregates are updated by sending the value of the selected state to *aggregators* provided by the *structuring level* and classified as unexploited. If the provided *aggregators* are classified as outdated, the earlier selected state is sent as well. The `initialiseCoordination` subtask initiates this communication, the `finaliseCoordination` subtasks completes it and the `organise` subtask realises the computation of the aggregation functions as defined in Algorithm 13 and 14 of the ASMA architecture.

The *coordination level* forms an overlay network between *aggregators* and *disseminators* linked with overlay links that have two possible semantic values: unexploited or outdated but not exploited. Therefore, the aggregation functions computed exclude overlay links from the *coordination level* that result in duplicate aggregation values (exploited *aggregators*). The aggregation memberships, the classification process, the selections of *aggregators* are all complexity that is hidden from the aggregation process of the *coordination level*. Adaptation strategies tune the aggregation process in favour of (i) discovering new selected states in the system (EXPLOITATION strategy) or (ii) updating the aggregates with the most recent selected states (UPDATE strategy). The *coordination level* has to only provide the classification *criteria* that trigger this optimisation and inform about changes in the selected state. Therefore, the *coordination level* remains agnostic about the details of the optimisation. The aggregation *criteria* define how the aggregates are provided to applications, e.g., periodic delivery or delivery when aggregates converge to the actual aggregate values by monitoring a minimum deviation threshold.

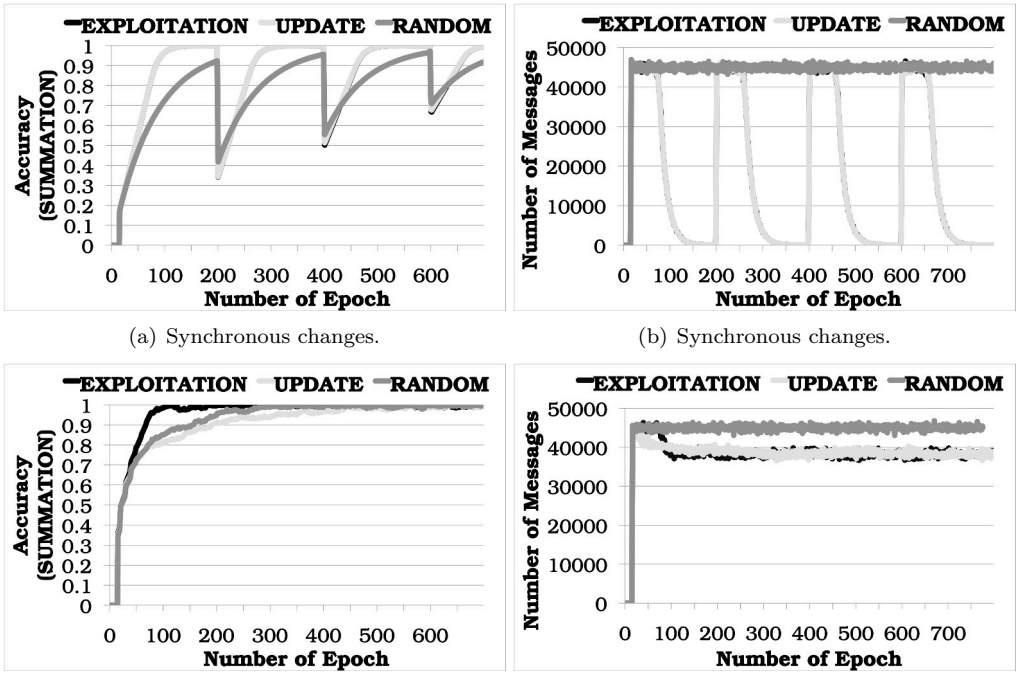
Figure 5.6 illustrates performance trade-offs between accuracy and communication cost for the adaptation strategies of DIAS. The accuracy measures how close the estimates of the aggregates, e.g., SUMMATION, is to the actual values [29]. Two scenarios are illustrated regarding how selected states change during runtime: (i) *synchronous* and (ii) *asynchronous* changes. In synchronous changes, the selected states of all peers in the network change simultaneously. In contrast, asynchronous changes occur arbitrary over time. In the illustrated experiments, synchronous changes occur every 200 epochs whereas in asynchronous changes, 420 selected states probabilistically change on average every 10 epochs.

Figure 5.6a and 5.6b show that after a synchronous change, accuracy drops dramatically and is restored to maximum within 100 epochs. This adaptation of aggregates causes a maximum of 45000 messages per epoch that drop to zero during the convergence period of 100 epochs. However, the communication cost of 45000 messages per epoch is constant for the RANDOM strategy.

Figure 5.6c and 5.6d show that despite the continuous changes of selected states every 10 epochs, DIAS is capable of maintaining a high accuracy after the initial convergence at system startup. However, a constant communication cost of 38000 messages per epoch is required to maintain this high accuracy that is yet lower than the 45000 messages per epoch of the RANDOM strategy.

DIAS can compute a wide range of aggregation functions at a performance comparable with the one shown in this section. In contrast to related methodologies reviewed in earlier work [29], DIAS does not require any architectural changes to compute a different aggregation function. It is because of the ASMA architectural modularity that the aggregation process is separated from routing. The two adaptation strategies provide a high reconfigurability in different networks settings. For example, during network scaling, EXPLOITATION is more effective than UPDATE. However, for a stable network with frequent changes, UPDATE is superior.

6. Comparison with Related Work. In earlier work [10, 11], the idea of 'open overlays' is introduced supported by a generic framework for overlay networks and their applications. This framework receives plug-



(a) Synchronous changes. (b) Synchronous changes.

(c) Asynchronous changes. (d) Asynchronous changes.

FIG. 5.6. *Performance trade-offs for the three adaptation strategies of DIAS:* EXPLOITATION, UPDATE *and* RANDOM *[29].*

in overlays defined by three components: 'Forwarding', 'state' and 'control'. These components are, in some aspects, similar to the three levels of ASMA. Applications are also introduced as plug-ins and are associated with possible overlay plug-ins that can support them. A plug-in can be positioned in the framework as independent or stacked with other plug-ins. Similarly with ASMA, top-down configurations are applied during deployment starting from the application plug-ins to the lower level network plug-ins. However, the number of possible combinations defined by the top-down configurations between the available overlay plug-ins can be large resulting in complex compositions.

Although dealing with overlay plug-ins is a generic, extensible and highly modular approach, the development of the three overlay components may be blended and cannot always be intuitive enough [10, 11]. Furthermore, the framework of overlay plug-ins does not define any high-level semantic of the component interactions. In contrast, ASMA provides a narrower defined context and objective for every self-organisation level. It also shows how these objectives are mutually supported via the exchange of *criteria* and *samples*.

Some related work [17, 20] focuses on supporting multiple overlays network capabilities as an IP-layer solution instead of a middleware solution that ASMA proposes for its realisations. OCALA [17] positions the 'overlay convergence' layer, built by an overlay-independent and an overlay-dependent component, under the transport layer. These layers provide a level of routing and lookup transparency between physical machines belonging to different overlay networks. However, there is a plethora of problems and open issues related to the support of existing IP-based applications, security, efficiency and access to overlay functions beyond routing. MOSAIC [20] is a declarative methodology for the composition of 'horizontal' (bridged via gateways) or 'vertical' (layered similarly to ASMA) overlay networks. Although this methodology provides a highly configurable and reasoning compositional environment for overlay networks, a large amount of information must be known a priori for each individual peer of the network. In addition, MOSAIC is highly dependent on a directory service that supports the composition process. It is unclear how changes to the directory service can be automatically reflected in the composed overlay networks. MOSAIC also faces the restrictions of an IP-layer solution similarly with OCALA. These approaches could in theory function complementary to ASMA overlay services for supporting communication between heterogeneous networks, e.g., wireless and wired networks.

iOverlay [18] provides an interface for building overlay networks and their applications. This interface is rather limited as it only supports overlay communication leaving excessive freedom to the developer. In comparison with ASMA, the main functionality of iOverlay corresponds to the *discovery level* of ASMA. A similar approach with iOverlay is followed by MACEDON [33]. Opus [4] is based on a backbone service to optimise the resource allocation for different applications. Therefore, the scope of this approach is limited compared to ASMA and the other approaches illustrated in this section. A multi-level economic framework for Grid services and resource allocation is earlier introduced [16]. Self-organisation is engaged for the discovery of agents that negotiate for resources. An overlay abstraction is provided to the agents. The system is designed based on web service technologies and therefore some of its components remain centralised. In contrast, ASMA introduces multiple self-organisation levels for system discovery, structuring and coordination without centralised components but in a collective fashion.

7. Conclusion and Future Work. This paper shows that collective adaptive systems can be designed and prototyped to provide modular and reconfigurable capabilities of a broad application scope: the overlay services. This paper contributes the ASMA conceptual architecture that guides realisations of complex overlay services via a few lines of high-level algorithmic expressions. The realisation of two overlay services according to ASMA together with the earlier experimental results illustrated in the new context of this paper empirically justify their higher abstraction, modularity and reconfigurability.

AETOS builds and maintains collectively different tree topologies with different topological properties that meet several application requirements. Topological reconfigurations in the self-organisation process are exclusively managed by plugged-in adaptation strategies that can be dynamically combined during runtime to improve performance under various scenarios such as node failures or network scaling. Similarly, DIAS computes almost any aggregation function that receives for input dynamically changing values distributed in a network. Adaptation strategies configure aggregation to compute in priority uncounted or outdated values depending on various network scenarios in which new nodes enter the network or regularly change their values. In both overlay services, the three levels of ASMA provide an intuitive and structured pathway to dissect the complex functionality of these systems in stand-alone, modular and reconfigurable subsystems. Although it is inevitable that this generic distributed computing approach has an impact on performance, e.g., high communication cost, overlay services allow reconfigurability with trade-offs. For example, the eight adaptation strategies of AETOS provide a spectrum of choices between high or low communication cost and performance. The dynamic adoption of adaptation strategies provide the option to explore this spectrum to improve the overall cost-effectiveness of overlay services.

Future work concerns the further realisation of overlay services according to ASMA. Usability case-studies and development scenarios shall strengthen the potential of a new distributed computing paradigm for collective adaptive systems based on overlay services.

Acknowledgements. The author would like to recall all people acknowledged in the preface of his PhD thesis [25] for making this research possible.

REFERENCES

[1] T. Anderson, L. Peterson, S. Shenker, and J. Turner, *Overcoming the Internet Impasse through Virtualization*, Computer, 38 (2005), pp. 34–41.

[2] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, *Model-Based Performance Prediction in Software Development: A Survey*, IEEE Transactions on Software Engineering, 30 (2004), pp. 295–310.

[3] A. Boulis, S. Ganeriwal, and M. B. Srivastava, *Aggregation in sensor networks: an energy-accuracy trade-off*, Ad Hoc Networks, 1 (2003), pp. 317–331.

[4] R. Braynard, D. Kostic, A. Rodriguez, J. Chase, and A. Vahdat, *Opus: an Overlay Peer Utility Service*, in Proceedings of Open Architectures and Network Programming, OPENARCH 2002, Los Alamitos, CA, USA, Jan. 2002, IEEE, pp. 167–178.

[5] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen, *Deployment Issues for the IP Multicast Service and Architecture*, IEEE Network, 14 (2000), pp. 78–88.

[6] J. Fan and M. H. Ammar, *Dynamic Topology Configuration in Service Overlay Networks: A Study of Reconfiguration Policies*, in Proceedings of the 25th International Conference on Computer Communications, INFOCOM 2006, Los Alamitos, CA, USA, Apr. 2006, IEEE, pp. 1–12.

[7] W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer, *ProtoPeer: A P2P Toolkit Bridging the Gap Between Simulation and Live Deployement*, in Proceedings of the Second International Conference on Simulation Tools and Techniques, ICST 2009, Gent, Belgium, Mar. 2009, ACM, pp. 1–9.
[8] C. Gkantsidis, M. Mihail, and S. Saberi, *Random walks in peer-to-peer networks: Algorithms and evaluation*, Performance Evaluation, 63 (2006), pp. 241–263.
[9] A. González-Beltrán, P. Milligan, and P. Sage, *Range queries over skip tree graphs*, Computer Communications, 31 (2008), pp. 358–374.
[10] P. Grace, G. Coulson, G. S. Blair, L. Mathy, W. Kit Yeung, W. Cai, D. A. Duce, and C. S. Cooper, *GRIDKIT: Pluggable Overlay Networks for Grid Computing.*, in Proceedings of Cooperative Information Systems, CoopIS 2004, vol. 3291 of Lecture Notes in Computer Science, Heidelberg, Oct. 2004, Springer-Verlag Berlin, pp. 1463–1481.
[11] P. Grace, D. Hughes, B. Porter, G. S. Blair, G. Coulson, and F. Taiani, *Experiences with Open Overlays: A Middleware Approach to Network Heterogeneity*, in Proceedings of the 3rd European Conference on Computer Systems, SIGOPS/EuroSys 2008, New York, USA, Apr. 2008, ACM, pp. 123–136.
[12] D. Haage, R. Holz, H. Niedermayer, and P. Laskov, *CLIO - A Cross-Layer Information Service for Overlay Network Optimization*, in Kommunikation in Verteilten Systemen (KiVS), Informatik aktuell, Heidelberg, Mar. 2009, Springer-Verlag Berlin, pp. 279–284.
[13] M. Jelasity, A. Montresor, and O. Babaoglu, *T-Man: Gossip-based fast overlay topology construction*, Computer Networks, 53 (2009), pp. 2321–2339.
[14] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, *Gossip-based Peer Sampling*, ACM Transactions on Computer Systems, 25 (2007).
[15] S. Jiang, L. Guo, and X. Zhang, *LightFlood: an Efficient Flooding Scheme for File Search in Unstructured Peer-to-Peer Systems*, in Proceedings of the 2003 International Conference on Parallel Processing, ICPP 2003, Los Alamitos, CA, USA, Oct. 2003, IEEE, pp. 627–635.
[16] L. Joita, O. F. Rana, P. Chacín, I. Chao, F. Freitag, L. Navarro, and O. Ardaiz, *Application deployment using catallactic Grid middleware*, in Proceedings of the 3rd International Workshop on Middleware for Grid Computing, MGC 2005, New York, USA, Nov. 2005, ACM Press, pp. 1–6.
[17] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle, *OCALA: An Architecture for Supporting Legacy Applications over Overlays*, in Proceedings of the 3rd International Conference on Networked Systems Design & Implementation, NSDI 2006, Berkeley, CA, USA, May 2006, USENIX Association, p. 20.
[18] B. Li, J. Guo, and M. Wang, *iOverlay: A Lightweight Middleware Infrastructure for Overlay Application Implementations*, in Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware, Middleware 2004, vol. 3231 of Lecture Notes in Computer Science, New York, USA, Oct. 2004, Springer-Verlag New York, pp. 135–154.
[19] S. P. Mahambre, M. Kumar S.D., and U. Bellur, *A Taxonomy of QoS-Aware, Adaptive Event-Dissemination Middleware*, IEEE Internet Computing, 11 (2007), pp. 35–44.
[20] Y. Mao, B. T. Loo, Z. Ives, and J. M. Smith, *MOSAIC: Unified Declarative Platform for Dynamic Overlay Composition*, in Proceedings of the International Conference on Emerging Networking Experiments and Technologies, CONEXT 2008, New York, USA, Dec. 2008, ACM Press, pp. 1–12.
[21] R. Matei, A. Iamnitchi, and P. Foster, *Mapping the Gnutella network*, IEEE Internet Computing, 6 (2002), pp. 50–57.
[22] K. Nahrstedt, R. Chang, and C. Ward, *QoS-Assured Service Composition in Managed Service Overlay Networks*, in Proceedings of the 23rd International Conference on Distributed Computing Systems, ICDCS 2003, Los Alamitos, CA, USA, May 2003, IEEE, pp. 194–201.
[23] A. L. Nelson, G. J. Barlow, and L. Doitsidis, *Fitness functions in evolutionary robotics: A survey and analysis*, Robotics and Autonomous Systems, 57 (2009), pp. 345–370.
[24] J. O'Madadhain, D. Fisher, P. Smyth, S. White, and Y.-B. Boey, *Analysis and Visualization of Network Data using JUNG*, Journal of Statistical Software, 10 (2005), pp. 1–35.
[25] E. Pournaras, *Multi-level Reconfigurable Self-organization in Overlay Services*, PhD thesis, Delft University of Technology, March 2013.
[26] E. Pournaras, G. Exarchakos, and N. Antonopoulos, *Load-driven neighbourhood reconfiguration of Gnutella overlay*, Computer Communications, 31 (2008), pp. 3030–3039.
[27] E. Pournaras, M. Warnier, and F. M. T. Brazier, *Adaptation Strategies for Self-management of Tree Overlay Networks*, in Proceedings of the 11th IEEE/ACM International Conference on Grid Computing, Grid 2010, Los Alamitos, CA, USA, Oct. 2010, IEEE, pp. 401–409.
[28] E. Pournaras, M. Warnier, and F. M. T. Brazier, *Local Agent-based Self-stabilisation in Global Resource Utilisation*, International Journal of Autonomic Computing, 1 (2010), pp. 350 – 373.
[29] E. Pournaras, M. Warnier, and F. M. T. Brazier, *A generic and adaptive aggregation service for large-scale decentralized networks*, Complex Adaptive Systems Modeling, 1 (2013).
[30] E. Pournaras, M. Warnier, and F. M. T. Brazier, *Adaptive self-organization in distributed tree topologies*, International Journal of Distributed Systems and Technologies, 5 (2014).
[31] E. Pournaras, M. Warnier, and F. M. T. Brazier, *Peer-to-peer aggregation for dynamic adjustments in power demand*, Peer-to-Peer Networking and Applications, 8 (2014), pp. 189 – 202.
[32] M. Puviani, G. Cabri, and F. Zambonelli, *A taxonomy of architectural patterns for self-adaptive systems*, in Proceedings of the International C* Conference on Computer Science and Software Engineering, C3S2E '13, New York, NY, USA, 2013, ACM, pp. 77–85.
[33] A. Rodriguez, C. Killian, S. Bhat, D. Kostić, and A. Vahdat, *MACEDON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks*, in Proceedings of the 1st International Conference on Networked Systems

Design and Implementation, NSDI 2004, Berkeley, CA, USA, Mar. 2004, USENIX Association, pp. 267–280.
[34] G. Tan, S. A. Jarvis, X. Chen, D. P. Spooner, and G. R. Nudd, *Performance Analysis and Improvement of Overlay Construction for Peer-to-Peer Live Media Streaming*, Simulation, 82 (2005), pp. 169–178.
[35] S. Tang, E. Jaho, I. Stavrakakis, I. Koukoutsidis, and P. V. Mieghem, *Modeling gossip-based content dissemination and search in distributed networking*, Computer Communications, 34 (2011), pp. 765–779.
[36] P. Van Mieghem, J. Omic, and R. Kooij, *Virus spread in networks*, IEEE/ACM Transactions on Networking, 17 (2009), pp. 1–14.
[37] R. Van Renesse, K. P. Birman, and W. Vogels, *Astrolabe: A Robust and Scalable Technology For Distributed System Monitoring, Management, and Data Mining*, ACM Transactions on Computer Systems, 21 (2003), pp. 164–206.
[38] J. Yuh-Jzer, F. Chien-Tse, and Y. Li-Wei, *Keyword Search in DHT-Based Peer-to-Peer Networks*, in Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, ICDCS 2005, Los Alamitos, CA, USA, Jan. 2005, IEEE, pp. 339–348.
[39] D. Zhenhai, Z. Zhi-Li, and Y. Hou, *Service Overlay Networks: SLAs, QoS and Bandwidth Provisioning*, IEEE/ACM Transactions on Networking, 11 (2003), pp. 870–883.
[40] H. Zhuge and L. Feng, *Distributed Suffix Tree Overlay for Peer-to-Peer Search*, IEEE Transactions on Knowledge and Data Engineering, 20 (2008), pp. 276–285.
[41] M. Wirsing, M. Hölzl, M. Tribastone, and F. Zambonelli, *ASCENS: Engineering Autonomic Service-Component Ensembles*, in Proceedings of the 10th International Symposium of Formal Methods for Components and Objects, FMCO 2011, (2013), Springer.
[42] M. Amoretti, *Introducing Artificial Evolution into Peer-to-Peer Networks with the Distributed Remodeling Framework*, Genetic Programming and Evolvable Machines, 2, (2013), pp 127–153, Springer.
[43] R. Bruni, A. Corradini, F. Gadducci, A. Lluch Lafuente, A. Vandin, *A Conceptual framework for adaptation*, in Proceedings of the 15th International Conference on the Fundamentals of Software Engineering, FASE 2012, Tallinn, Estonia, (2012), Springer.
[44] M. Amoretti, *Evolutionary strategies for ultra-large-scale autonomic systems*, Information Sciences, 274. (2014), pp. 1–16.

Edited by: Giacomo Cabri and Emma Hart
Received: Dec 15, 2014
Accepted: Jul 15, 2015