

M2M Architecture with Node and Topology Abstractions

INGE GRØNBÆK



Inge Grønbaek is Senior Adviser in Telenor Group Business Development and Research

Wireless M2M ad-hoc type networks are typically resource limited due to the restricted capabilities of constituent objects (ie. nodes). There may be limitations in transmission range and capacity, power availability, spatial coverage and location, etc. These limitations may be alleviated through abstractions implemented by cooperating objects. Such abstractions span a range of diverse approaches, each suited to specific ambient conditions and applications. This work proposes *management service ontology*, defining an API, for easy access to and use of the following main types of regional abstractions: Abstract regions, *Virtual Nodes* (VNs) for trajectory control and *virtual graphs* and *topologies* by emulation. The API extends the power of regions by allowing applications development to take place without considering the characteristics of the underlying network topology and ambient conditions. Management supports adoption of the most efficient virtualization depending on the environment of deployment. The *Virtual Node Layer* (VNL) architecture is developed to support the region and topology abstraction implemented as a VN application or as a Physical Node (PN) application. Implementing applications on top of the VNL provides the extra reliability and resilience which may be needed in certain environments, eg. for wireless sensor networks. This represents a major simplification which provides flexibility for optimisations to the operational environment and underlying technology without modifications to the implemented algorithms.

1 Introduction

In the architecture proposed in [1] the functionality is offered to Connected Objects (COs) via an API [2], [3]. Functionality is either offered on a peer-to-peer (P2P) basis or enabled by a minor set of new generic functional entities. These include a gateway and an anchor point entity class. The gateway (GW) can be instantiated for interconnect of a rich variety of COs, including layer two proprietary COs, ie. COs using the API with limited capabilities locally in the device network, and relying on the higher layer protocols of the GW for interconnect beyond the reach of the layer 2 LAN. The architecture is designed also to support native General Packet Radio Service (GPRS) devices. The anchor point entity handles global mobility management and mobile M:N multicast. Additionally, these new entities support presence and location based services. Locations of COs can be found, and the set of COs at a specified location may be identified. Privacy is also accommodated.

The architecture is functionally layered, with protocols and entities identified for each layer of the well known ISO OSI reference model. Diverse capabilities and protocol stacks may be supported by resolving the CO profile from the CO identity. Deployed Internet protocols and protocols under development by the Internet Engineering Taskforce (IETF) are adopted, and the new functionality may be based on current Internet router architecture with minor augmentations.

The critical element for the support of ubiquitous services is semantic and physical interoperability. An Application Programming Interface (API) at selected layers of the protocol stack (eg. as depicted for the Application and what is defined here as the Internet layer in Figure 1) is key to support both interoperability and service ubiquity.

Web services protocols are proposed for the backbone side of the architecture, while simpler protocol stacks and representations may be applied internal to device and sensor networks. Ontologies for M2M service interactions are logically allocated to the presentation layer of the proposed architecture. This defines the vocabulary for different application areas and services. The interconnect architecture offers interoperability at the Internet layer (Figure 1) via network elements like gateways and servers (Figure 2), while COs communicating beyond device networks interoperate above the Internet layer applying higher layer protocols, including the Presentation ontology (for semantic compatibility) and any available application component building blocks, all accessed via the application API.

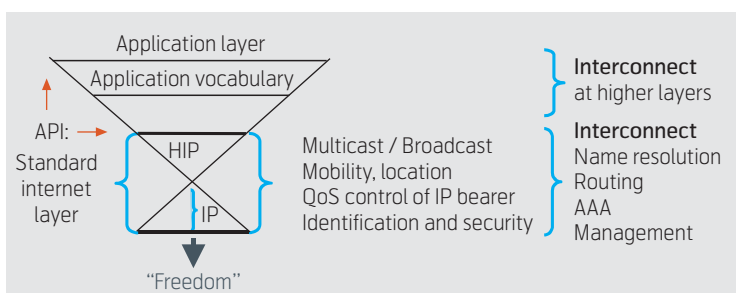


Figure 1 Internet layer with API

In device networks there may be limitations in transmission range and capacity, in power availability, in spatial coverage and location, etc. These limitations may be alleviated by abstractions implemented by cooperating objects.

Such abstractions span a range of diverse approaches each suited to specific ambient conditions and applications. This work proposes an API defined as ontology, as a supplement to the M2M architecture, for easy access to and use of the following main types of abstractions:

- 1 Abstract regions
- 2 Virtual nodes for trajectory control
- 3 Virtual graphs and topologies of abstract regions

The rest of the paper describes the abstractions with service API and concludes with remarks and recommendations.

1.1 Topology and Network Elements

The architecture of an M2M network [1] may include a backbone network, separate device networks, Connected Objects (COs) or devices, gateways and servers. The proposed network for the Internet of Things (IoT) comprises two logically distinct but closely coupled network domains, ie. the backbone and the device network domains. Both domains may be serving devices directly. Figure 2 provides a component-view of the architecture.

The following is a brief description of the functional components of the architecture.

Connected Objects: The Connected Objects (COs) are the end systems or devices in the device network.

Device Networks: The Device Networks connect clusters of COs and must be flexible in technology and topology. A limited set of interfaces need to be standardised for interconnect with and via the backbone network. Device networks and their devices (eg. sensors and actuators) are expected to develop continuously for a longer period, and their technology basis is an important topic for continued research. There is a need to integrate simple low-end devices, eg. with limitations in functionality and power supply. Such elements are interfaced efficiently through a flexible gateway architecture proposed in [1].

The use of the Host Identity Protocol (HIP) [4] is proposed for implementing the 'Internet layer'. HIP defines a new name space between the network and transport layers (Figure 1 and Figure 3). HIP provides upper layers with mobility, multihoming, NAT (Network Address Translation) traversal, and security functionality. HIP implements the so-called identifier/locator (ID/locator) split, which implies that IP addresses are only used as locators, not as host identifiers. This split makes HIP a suitable protocol to build overlay networks that implement identifier-based overlay routing over IP networks, which in turn implement locator-based routing.

Architecturally, HIP can be considered to create a new thin 'waist' layer (ie. the Internet layer) on top of the IPv4 and IPv6 networks, see Figure 1. The HIP layer itself consists of the HIP signalling protocol and

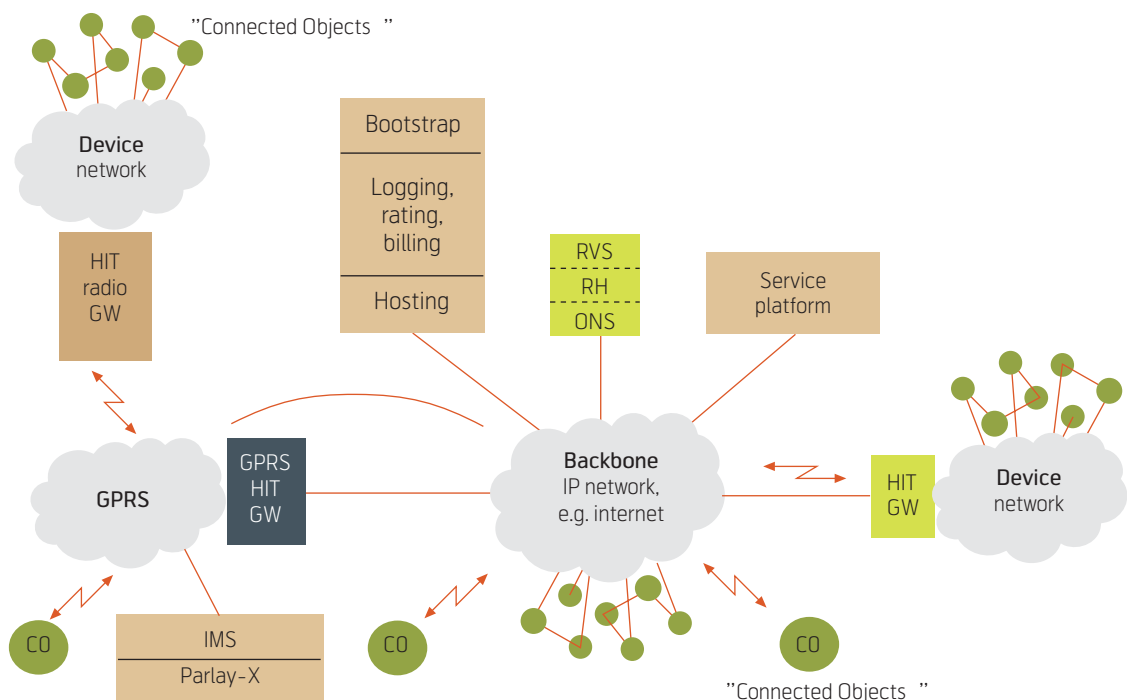


Figure 2 Components and their Interrelations

one or more data transport protocols. The HIP signalling packets and the data transport packets can take different routes. HIP signalling packets are typically first routed through a rendezvous server (RVS in Figure 2) and then directly (if possible), while the data transport packets are typically routed only directly between the end points.

In order to make HIP work through the traditional IPv4 socket API to support legacy applications, the HIP module passes an LSI (Local Scope Identifier), instead of a regular IPv4 address, to the legacy IPv4 application. The LSI looks like an IPv4 address, but is locally bound to a Host Identity Tag (HIT). That is, when the legacy application uses the LSI in a socket call, the HIP module intercepts it and replaces the LSI with its corresponding HIT. Therefore, LSIs always have local scope. They do not have any meaning outside the host running the application.

Backbone Network: The Backbone Network offers ubiquitous interconnect for services at the Internet layer, while COs interconnect at the Application layer, ie. the higher layer protocols are transparent to the Internet layer. All interconnect with non backbone compliant architectures is carried out at the rim of the backbone, and this effectively eliminates the need for the N square interconnect arrangements and gateways for N different technologies (eg. service specific networks).

Host Identity Tag (HIT) Gateway: The Host Identity Tag (HIT) Gateway is based on the Host Identity Protocol [4] and allows global addressing of COs while avoiding rapid depletion of the IPv4 address-space. This is achieved by allocating a single public IP-address to a potentially large group of COs under control of a single HIT gateway. This is the address of the HIT gateway. The HIT gateway shall keep track of the location of all COs under its control. Each gateway shall be allocated a coverage area allowing identification of objects within that area. Each gateway shall furthermore keep track of all its physical neighbours to allow extended area search for COs. The communication scheme on the device network side of the gateway (Figure 3) can be based on local light weight protocols, eg. at the link layer. The layers denoted 'transparent' may or may not be instantiated at the device side of the gateway.

The **HIT Radio Gateway** is functionally equivalent to the HIT gateway except for applying a radio interface (eg. GSM, EDGE, UTRAN) for access to the GPRS network.

Rendezvous Server (RVS): The basic functionality of the Rendezvous Server (RVS) is to offer mobility

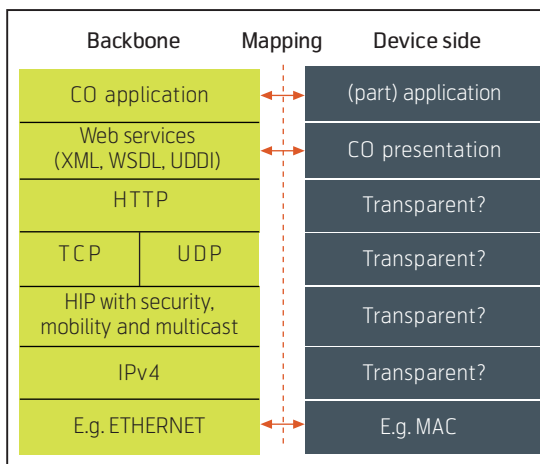


Figure 3 HIT Gateway Architecture

anchoring, ie. maintenance of the HIT to address bindings.

Resolution Handler (RH): The Resolution Handler (RH) is an RVS extension offering generic name resolution from a flat namespace (eg. HIT to address resolution).

Service Platform: The M2M service architecture [5], as shown in Figure 4 has the following components:

- M2M devices network: Heterogeneous wireless/wired small devices connected using different network protocols, some IP based (interconnected through routers) and others non-IP (requiring gateways).
- Backbone interface: Devices and device networks will be connected though different access technologies (xDSL, Ethernet, satellite, GPRS, CDMA, GSM, HSDPA, etc.).
- Backbone Network: Infrastructure network, like Internet, NGN or GPRS/3G.
- Service Platform: M2M platform providing a set of functionalities that will allow efficient service development, deployment and operation.

Each of the building blocks shall have standard interfaces. At the lower layer, the M2M network protocols should be open and standardized. At the network layer the interface (standard interface 4) should support the functionality of a standard Internet API, so that different M2M networks supporting the API will be interoperable. At the service layer, a standard API should be provided to ensure application interoperability among heterogeneous M2M networks adopting different lower layer protocols. This API should support the functionality of Interface 5.

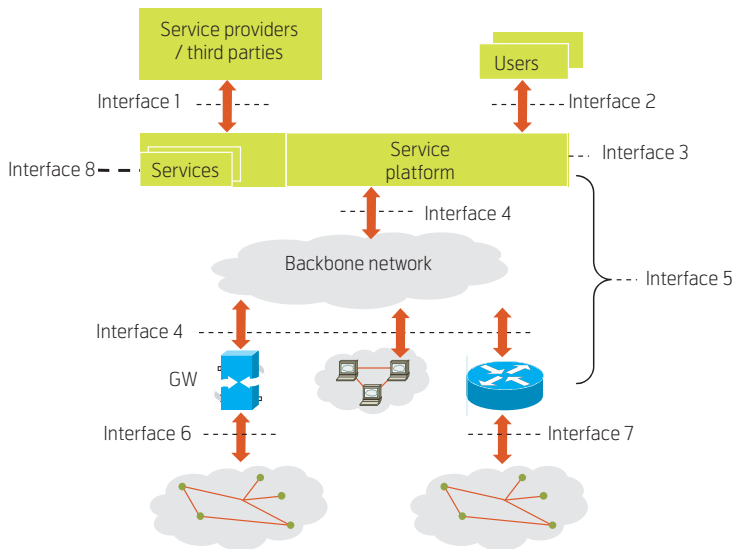


Figure 4 Service architecture

Interface 1 is the interface between the platform and external service providers. The API for this interface is provided for the application developers and service providers.

In addition the application developers can use the API offered by the platform modules and other 3rd party modules that are made available. This interface includes the functionality of Interface 2.

Interface 2 is the interface between the platform and the customer. The interface may be supported on leased lines or on the Internet. The default application protocol is XML Web Services. Other application protocols may be chosen if required by the customer. Interface 2 shall support the following functions:

- Information exchange between the platform and the customer;
- Downloading of software from the customer;
- State-of-the-art firewalls, antivirus programs and spam filters updated at regular intervals;
- Intrusion detection management and reporting.

Tight access management and security functions such as two-way device authentication, message authentication and integrity management, encryption of IP connection, encryption of information, and event logging.

Interface 3 is a set of interfaces supporting additional functionality. Such functions are:

- Installation support and initiation of automatic testing (important for high volume end-system deployments eg. for AMR);
- Access to remote databases for data storage and logging;

- Access to repositories for downloading of software;
- Remote operation and management of platform.

The functionality may include cooperation with other operators, eg. mobile providers for information security, load management in the access network, and QoS information such as outage of base stations, reduced traffic capacity, and other problems.

Interface 4 is the standard interface towards the backbone IP network (ie. the Internet layer in Figure 1). It includes functionality belonging to the three lowest layers of the ISO/OSI protocol stack, ie. the Internet protocol stack. Enhanced by the functionality of the HIP [4] it should provide all the functionality required for interconnect as listed on the right hand side of Figure 1, and additionally provide identification of location and QoS control.

Interface 5 is an interface between the service platform CO and the device COs, ie. a protocol stack as depicted on the backbone side of the gateway in Figure 3. The 3 lowest layers are identical to Interface 4.

The higher layers of the interface support the application layer protocols, (eg. XML Web Services, Remote Procedure Call), as depicted by the higher layers of the Backbone side of the GW in Figure 3. Characteristics of Interface 5 as offered via the API and the related platform functionality are:

- Establishment of connections from the platform to individual COs using individual identification or identification of groups of COs using either multicast or broadcast. In cases where this connection cannot be established directly from the platform to the CO (eg. as in GPRS), the platform must be able to initiate the call using alternative methods such as SMS.
- Connecting to COs may require execution of scheduling algorithms in order to avoid overloading the communication systems. In case of multicast and broadcast calls requiring responses from individual COs, the scheduling must take place in the COs based on randomization over a given time interval. The size of this interval may be contained as a parameter in the call from the platform or be a system parameter included in the COs during production, instantiation or installation.
- The platform must be ready to receive a call from a CO at any time. Calls may be processed in accordance with priority levels. Calls requiring confirmation should be answered in due time.

- Management of identification schemes including HIT, IP addresses, electronic product codes (EPC), and possibly also mobile identities and telephone addresses to individual COs. Incorporation of a global identity-scheme for M2M communication is an important standards issue.

The functions of the communication system are hidden from the processing in the platform by the API.

Interface 6 is proprietary and/or application specific, and may include functionality and protocols from all or only a few layers of the ISO/OSI stack.

Interface 7 may be identical to Interface 4. However, it may be optimized for M2M applications according to specifications from the IETF (eg. from the working groups 6lowpan and Roll).

Interface 8 supports trade of service components between service platforms, eg. it can be applied for access to IMS/Parlay-X functionality (Figure 2).

Support Features: The support features may include bootstrap, logging, rating, micro-payment, billing and hosting.

1.2 API Services and Layering

The APIs (eg. at the Internet layer in Figure 1) provide application portability and harmonised access to common lower layer functionality. At the same time, they separate the application from the mechanisms and technology used for communication, facilitating independent innovation of applications, protocols and infrastructure. This is a benefit of the well known principle of layering. The API furthermore describes capabilities and services between objects. Capabilities and services may freely be allocated to end systems (COs) or to servers. This enables functional allocations at the discretion of the developer. The same API may be used for network centric services and for P2P services. This enables services and service components to move, ie. the architecture is agnostic to functional allocation and location.

The actual CO protocol (eg. monitoring or remote control) is carried as payload by the user plane service elements of the API. The architecture allows new applications and protocols to be defined without changing the basic API or its primitives (ie. methods).

The service logically provided between COs, via the service API (Figure 5), shall be flexible in also offering subsets of the functionality. The idea, in particular applicable to device networks, is for the implementation to apply the simplest and most efficient protocol stack meeting the service requirements, with

no or minimum processing and transmission overhead. Figure 5 shows aggregation of functionality from the set of service layers and protocol entities in the protocol stack. Any of the layers shown may be functionally transparent, depending on the protocol stack in use for a given object. Flexibility is enabled by resolution of the CO characteristics, including protocol stack profile, from the CO identifier.

In local implementations it shall also be possible to support the network service API without including the basic IP bearer, eg. by providing the service directly above the link layer. Such a sub-IP approach will however require a gateway arrangement (Figure 3) to allow communication services to extend the local area beyond the reach of the applied link layer protocol.

1.3 Service Primitives at each Layer

The functionality of the API service offered to COs is aggregated from the services offered by the sub-layers shown in Figure 5. The description adheres to the well known ISO OSI reference model.

The aggregated IP bearer enhances the basic IP bearer service as will be offered by what is here called the Internet layer.

Management functions are in general applicable to all layers and across the layers.

The following service elements are defined as part of the API, but not presented here: Micro-Payment, Storage and retrieval, Presentation service, Session service, Transport service, CO presence, Location & status, Mobility, Basic IP bearer. A more complete overview of API services is given in [2] and [3].

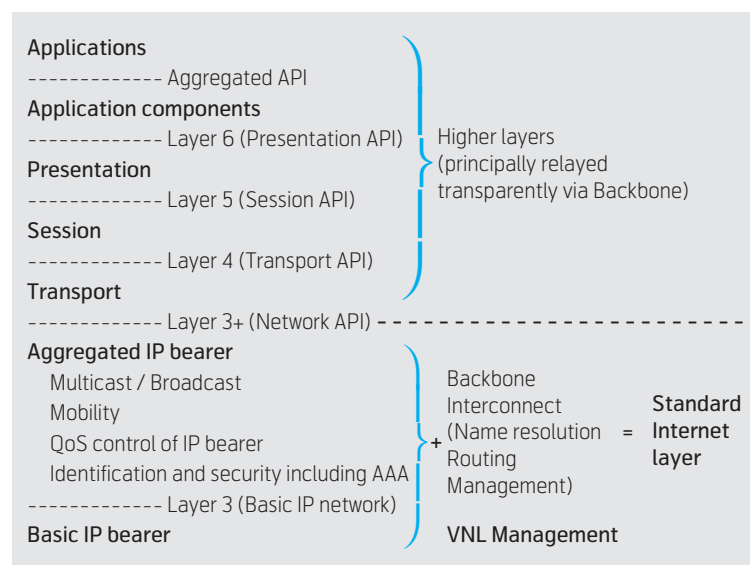


Figure 5 API and Service Aggregation

2 Device Network Design Approach

The implicit assumption [6] behind much of device and sensor network design and research is that the design of given protocols and algorithms are somehow independent of the nature of the wired or Wireless Sensor Network (WSN) applications, and that there is little or no variability across these applications. Such an assumption is clearly false [6], but tempting since there are many obvious benefits of generic solutions. However, applications vary widely on several aspects. The major shortcoming of WSN protocol research and design is lack of protocol evaluation. This involves considering a set of parameters including: (1) the number of nodes, (2) the density of nodes, (3) the pattern of node deployment, (4) the availability of power, (5) the kind of antennas in use, (6) the resultant expected number of hops in the network, (7) the required or desired network topology, (8) the traffic pattern, (9) the data generation rate, (10) the location of application level processing and data fusion, etc. The bottom-up approach proposed in [6] advocates design of protocols for specific scenarios with a secondary search for generality across the

specifics. This is a valid approach; however, the number of possible permutations of the parameters above, describing only a fraction of the involved design space, is so extensive that the approach seems unrealistic as a one-step endeavour. The approach advocated here is therefore to create a flexible framework for use, trial and evaluation of promising solutions for selected areas of the design-space. This flexibility is reflected in the gateway design as depicted in Figure 5, which will enable interoperability between varying designs of devices and device networks. In order to forward this approach and also to cover the basic functionality of the device network we design an abstract configurable service which may be used by system developers to abstract away the details of diverse operational conditions and underlying network topologies in their implementations by initially focusing on application oriented features and functionality. The proposed features are accessed via an efficient API defined in the form of ontology. A ‘RESTful’ [7] approach is adopted maintaining orthogonality in identification, methods and data definitions, ie. new data structures are described by ontology.

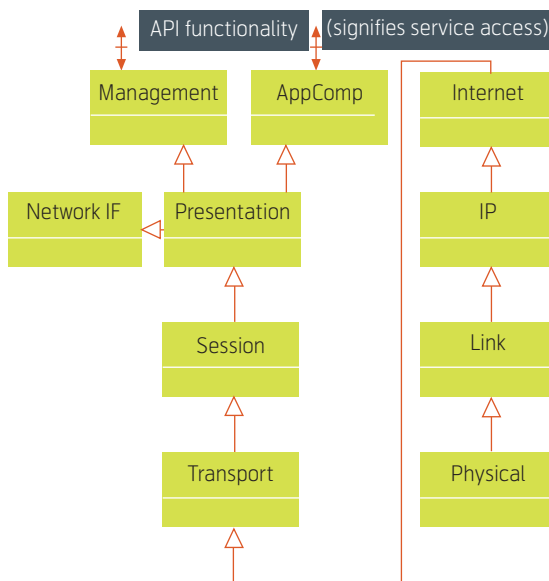


Figure 6 Service Ontology

The service ontology provides a service classification and a specification scheme for service functionality. There is a service class for each layer of the protocol stack (Figure 6). Each service class is associated with its corresponding service primitives or attributes (slots). All basic service primitives shall offer a container for carrying data of a separately defined type. This creates an environment where clients and servers that encode their information the same way can work together by sharing ontology.

Figure 6 models the service ontology for the IoT. The class *NetworkIF* is a class shared with the network ontology. This ensures a common interpretation of the performance requirements across the network and service ontologies.

The following is a brief description of sample service primitives (from [2] and [3]) of the *Application Component* and the *Internet* classes from this *service ontology*.

Class Application Component (AppComp): The intention of the *Application Component* class is to predefine a small set of high level service primitives considered of value for service developers. However, most importantly it defines an extendable class which may be amended by the service development community for exchanging valuable proven service components and services. The advertisement and exchange of these services may be supported by the Universal Description, Discovery, and Integration (UDDI) registry. Table 1 provides example primitives for event reporting.

Primitives (Slots)	Parameters	Description
Event-Subscription-Send	(Registrar-CO-ID, Target-CO-ID, Parameters)	Event subscription at the event server
Event-Notification-Send	(Subscriber-CO-ID, Target-CO-ID, Parameters)	Event notification from the event server
Event-Report-Send	(Registrar-CO-ID, Parameters)	Event report from a CO

Table 1 Class Application Component, Event Service Primitives (Slots)

Primitives (Slots)	Parameters	Description
Send-ID	(OID, Data)	Sends to the identified CO (OID)
Receive-ID	(OID, Data)	Receives from identified CO (OID)
SA-Create	(OID, Profile, SA)	Security association (SA) creation
Send-SA	(SA, Data)	Sends to the specified SA
Receive-SA	(SA, Data)	Receives from specified SA
MC-Group-Open	(Group-ID, Profile)	Multicast group (MC). Profile specifies the characteristics, eg. security level
MC-Group-Close	(Group-ID)	Closes the multicast group
MC-Group-Join	(Group-ID)	The issuer of the primitive joins the group
MC-Group-Leave	(Group-ID)	The issuer of the primitive leaves the group
MM-Register	(OID)	The requestor of Mobility Management (MM) invokes primitive to activate MM
MM-End	(OID)	The issuing object transits to a stationary state
QoS-Set-Path	(Destination-OID, Profile)	Primitive for request of QoS control
QoS-Set-Path-Confirm	(Destination-OID, Profile, AoC)	Confirmation with Advice of Charge (AoC)

Table 2 Class Internet and its Service Primitives

Class Internet: The service provided by the *Internet* class aggregates the functionality from each of its subordinate layers into the service provided to the Transport, or directly to COs. Table 2 provides sample primitives which can also be applied by virtual nodes (VNs) as described in Section 3. It is interesting to note that the Multicast Group (MC-Group) service primitives are only special cases of primitives for management of a virtual spanning tree topology.

In the following sections the ontology is extended to cover the functionality of the generic VN Layer (VNL) of Figure 5. This requires amendments to the Management classes of the service ontology.

3 Virtual Node Layer

A solution to the problem of handling limitations in ad-hoc type of wireless device networks has been proposed in [8]. Virtual Node Layer (VNL) is an abstraction layer that provides a high-level, well-behaved network, emulated by the low-level network nodes. VNL can mask much of the uncertainty and dynamicity inherent in the setting of low power lossy networks, thereby simplifying the design of applications. Such an abstraction is crucial for efficient development of command and control applications such as using actuator equipped sensor nodes to control a factory production line where single node failure is not acceptable. This approach, in essence, reduces the challenge of handling the unpredictable nature of the network to the (one-time) task of carefully designing and analyzing the abstraction layer and its implementation.

The VNL mechanisms can be grouped into the following main categories [8-14], including their combinations:

- 1 Abstract regions
- 2 Virtual Nodes
- 3 Virtual Topologies

The VNL is very much confined to the Internet layer. The functionality is logically an overlay to the basic IP layer. The major functionality, consisting of neighbour discovery, topology discovery and maintenance, is classified here to be part of the Management and thereby residing within the management part of the Internet layer as shown in Figure 5. The communication oriented service of the Internet layer (Class Internet, Table 2) is kept agnostic of the region, node and topology abstractions. This implies that the application can be agnostic to the supporting topology, applied abstractions and involved dynamics. Only a management service is involved in setting up and maintaining the required VNL support. The following summarizes the abstractions.

3.1 Abstract Regions

A key design element is the abstract region as proposed in [14]. It is an architectural construct which may be used for large scale and wide distribution networks and offers a generic grouping and partitioning mechanism. Such a mechanism can provide increased functionality and management of unresolved problems in current networks. The rationale is that virtually any horizontal slice through the current Internet structure reveals a loosely coupled federation of sepa-

rately defined, operated, and managed entities. It is natural to think of each of these entities as existing in an abstract region of the network, with each abstract region having coherent internal technology and policies, and each abstract region managing its interactions with other abstract regions of the net according to some defined set of rules and policies. An abstract region is an entity that encapsulates and implements scoping, grouping, subdividing, and crossing boundaries of sets of entities. In network systems, these functions are used for a variety of purposes including scaling, heterogeneity, security, billing, performance, trust management, and so on. It is shown in [14] that we can separate mechanism from purpose, by providing a single highly optimized and reusable generic mechanism to serve a number of purposes. Virtualization proposed builds on [14] by exploiting the potential of abstract regions as the basic framework for further abstractions. Abstract regions support three groups of functionality: definition, membership, and boundary management. The features we have adopted for this work is shown in Table 3.

3.2 Virtual Nodes

The basic functionality of the VNL [8] is to present the application developer with two types of entities to program: predictable Virtual Nodes (VNs), and unpredictable Physical Nodes (PNs), which correspond to the physical nodes in the system. (See the VNL overview in Figure 7.) The dark green circles represent physical nodes, and the light green rectangles represent the VNs. PNs can fail unpredictably, and the set of PNs in a given region is unknown *a priori* and can change over time. VNs, on the other hand, are not intended to correspond directly to the underlying physical network. Rather, the VNs are identified with regions of the network, and may either remain in a fixed, known location or move in a controlled pattern [9] through the network. In Figure 7, each VN is within a region corresponding to the surrounding grid square. Each VN is emulated by a set

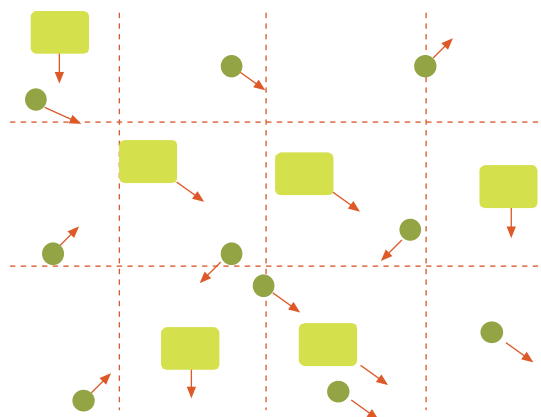


Figure 7 Virtual Node Layer (VNL)

of cooperating PNs running distributed algorithms including dynamic leader election for maintaining resilience against failing individual PNs. This emulation provides fault resilience allowing VNs to hide much of the dynamism of the underlying network. So instead of writing a distributed algorithm that must take into account the fact that the set of participating PNs is unknown and those participating might fail or leave during the lifetime of the VN, the developer can deploy a simple algorithm on the stable VNs. The complexity of dealing with the dynamic nature of the participants has been abstracted into the VNL service layer and dealt with once. The arrows in Figure 7 emphasize that PNs can be mobile while the VNs remain stationary or move in a predictable pattern.

Application examples are given in [8] including the implementation of a VNL virtual traffic light.

The mobility of the nodes also represents an opportunity. If the mobile nodes were moving in a useful way, it would be possible to take advantage of the motion to design algorithms that are even more efficient than those for static networks. Such distributed algorithms for VNs are proposed in [9]. The motion of a VN is determined in advance, and is known to the programs executing on the VNs. This contribution however, proposes additionally a method for downloading the virtual movement, eg. upon entering a region, effectuated by activating the Modify-invariants(...) primitive [14] described in Table 3. The motion of the VNs may be uncorrelated with the motion of the real nodes (PNs). Even if all the PNs are moving in one direction, the VNs can travel in the opposite direction. Consider, for example, an application to monitor traffic on a highway. Even though all the cars are moving in one direction, a VN could move in the opposite direction, eg. notifying oncoming cars of the traffic ahead.

The Mobile Point algorithm, defined in [9], allows PNs travelling near the location of a virtual node to emulate that VN. As the execution proceeds, the algorithm continually modifies the set of PNs for each VN so that the PNs always remain near the desired path of the VN. The work in [9] has a state machine approach, augmented to support joins, leaves, and recovery, to implementation. A VN may fail, however, as long as the path of the VN travels through dense areas of the network, the VN does not fail. If, however, the VN is directed to an empty spot, a failure may occur. The Mobile Point algorithm, however, allows the VN to recover to an initial state, if it again re-enters a populated area.

The underlying system model and algorithms defined in [9] consist of PNs moving in a bounded region of a

two-dimensional plane. Each PN is assigned a unique identifier. PNs may join and leave the system and may fail at any time. (Leaves are treated as failures.) PNs can move on any continuous path in the plane, with speed bounded by a constant. A Geosensor is a component of the environment that maintains the current location of each PN. It also maintains the current real time. The Geosensor can be implemented in real systems by a Global Positioning System (GPS) receiver, or other alternative, eg. for indoor usage.

3.3 Virtual Topologies in Abstract Regions

M2M sensor network applications are often expressed in terms of groups of nodes over which local sampling, computation, and communication occur. For example, tracking a moving object involves aggregating sensor readings from nodes near the object.

The goal of introducing virtual topologies [10] in abstract regions is to simplify application design by providing a set of programming primitives for sensor networks that abstract the details of low-level communication, data sharing, and collective operations. This offers a set of useful graphs (eg. trees, rings, neighbourhoods, etc.). Abstract regions may be defined in terms of radio connectivity, geographic location, or other properties of nodes. Abstract regions provide interfaces for identifying neighbouring nodes, sharing data among neighbours, and performing efficient reductions on shared variables. Abstract regions capture the inherent locality of communication and hide the details of data dissemination and aggregation within regions.

Example regions [10] covering the three examples shown in Figure 8 are:

- N-radio hop: Nodes within N radio hops
- N-radio hop with geographic filter: Nodes within N radio hops and distance d

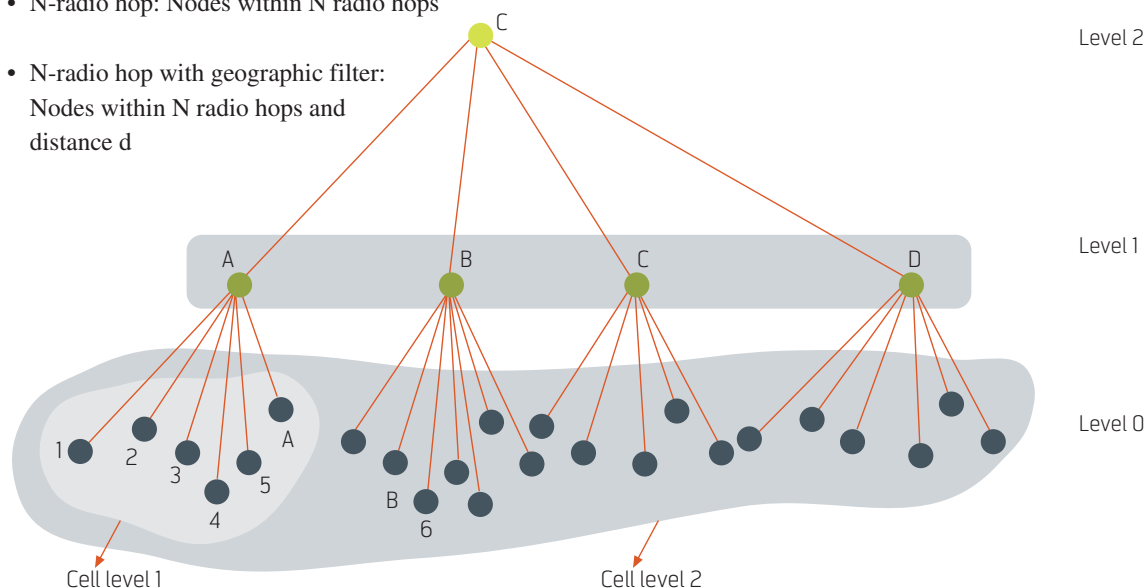


Figure 9 Virtual Topologies by Clustering

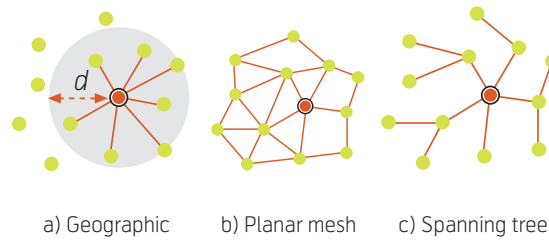


Figure 8 Abstract regions

- k-nearest neighbour: k nearest nodes within N radio hops
- k-best neighbour: k nodes within N radio hops with the highest link quality, as measured in fraction of packets dropped over some measurement interval
- Approximate planar mesh: A mesh with a small number (possibly zero) crossing edges
- Spanning tree: A spanning tree rooted at a single node, used for aggregating values over the entire network.

Many other topologies may be considered, but this presentation is limited to cover principles.

The architecture work in [10] addresses the abstraction requirement for multi-resolution data applications. The virtual topologies concept is complemented by the virtualization approach in [11] (Figure 9).

The hierarchy shown is a recursive organization of nodes into cells, cells into super-cells, and so on, based on an autonomous self-election of a subset of the nodes into drums, and iteratively drums self-elect-

ing to become higher level drums, and so on. Level-1 cells are also referred to as fundamental cells, as at this level, the cell is composed only of individual nodes. Figure 9 shows an example cell hierarchy where nodes 1, 2, 3, 4, 5, and A group together to form a cell (fundamental cell) with node A being the drum for the cell. Each of the drums in the network, namely nodes A, B, C, and D form a higher level cell with node C being the drum for that higher-level cell (called a super-cell). The tree structure can be application centric by only association nodes with specific capabilities, eg. nodes capable of measuring certain ambient conditions.

The functionality offered as Application Components based on the Cluster topology is described in Table 7 through Table 9.

The following describes the proposed service elements offered by VNs and Virtual topologies as part of the VNL. This is split into a management part, which mainly deals with discovery and configuration, and a communication part which uses the features offered by the Internet layer defined in Table 2.

4 VNL Abstractions and Management

The VNL is maintained by a dynamic and persistent discovery process which maintains the physical node topology for the virtual abstractions. Processes for this are described in [8-13], where [12-13] takes a theoretical approach in defining self-stabilizing Timed I/O automata.

Telecommunications Management Networks (TMNs) follow the OSI layered reference model. TMN provides an organization of management services in the form of a five layer hierarchy of services. The services provided by the TMN layers include:

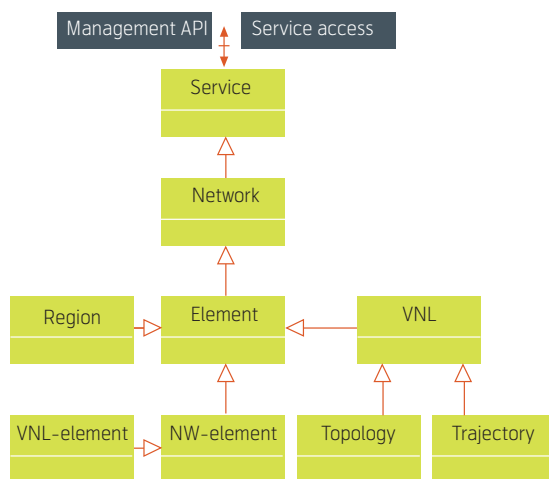


Figure 10 Management Service Ontology

- Business Management;
- Service Management;
- Network Management;
- Element Management – provides oversight and coordination of the services provided by groups of network elements;
- Network Element – provides agent services, mapping the physical aspects of the equipment into the TMN framework.

The management functionality of the VNL [15] is implemented at the Element Management and Network Element layer of TMN model, and the resulting functionality is offered to applications via the Service Management layer and its API as depicted in Figure 10. It may be noted here that Figure 10 represents an extension of the Management class of Figure 6. The functionality of each of the layers in the TMN model is represented by a service class (with its subclasses).

TMN layers have been extended here to include the VNL and the VNL-element (VNE), as displayed in Figure 8. The VNL is presented as a sub-class of the *Element* layer while VNE is presented as a subclass of the *Network Element* layer (NW-Element).

The *VNL* subclass of *Element* (*NW-Element*) builds on *Region* in providing *Topology* and *Trajectory* management for the VNL. The VNL class API is defined in Table 4. General management primitives, closing primitives, confirmations and error management are left out due to space restrictions. The parameters are defined as attributes for the *Topology* and *Trajectory* subclasses in Tables 5 and 6 respectively.

A VNL implementation [8] may be based on Reactive VNs, which are receive event-driven automata. That is, their operations are defined exclusively in terms of an *MsgReceived* handler (ie. a *Protocol* and *Service Data Unit* handler) which, upon being called with a received message, can transform the automata state and (potentially) return message(s) to be sent (eg. broadcast) in response. The input and output *Service Data Units* (ie. primitives) for the automata are defined in Table 1 through Table 6. Table 1 and Table 2 describe the application oriented primitives exchanged at the user plane, while Table 3 through Table 6 describe the management primitives used to control the underlying support of the user plane with the virtualization adopted for the actual operational environment and application.

The VNs fail if and only if their neighbourhood contains no physical nodes. It is assumed that messages might be lost, eg. due to collisions. The Reactive VN paradigm for the VNL was proposed in [8] as it simplifies the task of coding applications. That is, it is

easier to code a message handler than it is, for example, to code an arbitrary timed I/O automaton.

Each application (VN or PN) may also be implemented as receive event-driven automata using the service primitives defined here. Applications may be run natively on the PN, eg. when operating in a benign environment, or may apply the VNL when required for the application to perform eg. in a hostile or unstable environment. This implies that no additional overhead needs to be introduced when not explicitly required.

4.1 Policy Areas and Parameterization

This paragraph defines areas where policy control is required for device and sensor network applications. Furthermore the mechanism for binding these policy areas to the management classes are described as invariants for class Region and class VNL (Table 3 and Table 4).

4.1.1 Policy Areas

The following define the set of areas where policies are needed for parameterization of the VNL for adaptation to environmental conditions and operational requirements.

Resource Constrained Operations: VNL must support parameter constrained operations where the parameters are controllable resources like CPU, memory size, battery level, etc.

Energy-awareness and Power Management: VNL must be aware of the battery levels of the sensor nodes and seek to balance the energy consumption of the whole network.

However, not all the sensors, Electronic Control Units (ECUs), and actuators are operated via battery. Therefore, the system must save power consumption for these un-powered sensors, actuators, and ECUs. Powered devices should assist the un-powered devices or take care of more functionalities than un-powered devices. This may be coupled with Radio Resource Management.

Quality of Service (QoS): For mission-critical applications, support of QoS is mandatory. The following service parameters are relevant in a resource-constrained network:

- Data bandwidth – the bandwidth might be allocated permanently or for a period of time to a specific flow. Some flows may also share bandwidth in a best effort fashion.

- Latency – the time taken for the data to transit the network from the source to the destination. This may be expressed in terms of a deadline for delivery.
- Transmission phase – process applications can be synchronized through coordinated transmissions.
- Precedence and revocation priority – Networks may have limited resources that can vary with time. This means the system can become fully subscribed or even over-subscribed. System policies determine how resources are allocated when resources are over-subscribed. The choices are blocking and graceful degradation.
- Transmission priority – the means by which limited resources within devices are allocated across multiple services. For transmissions, a device has to select which packet in its queue will be sent at the next transmission opportunity. Packet priority is used as one criterion for selecting the next packet. For reception, a device has to decide how to store a received packet. The devices are memory constrained and receive buffers may become full. Packet priority is used to select which packets are stored or discarded.
- Reliability – Data provided for further processing must be transported reliably as if one part of the whole data set is lost, the entire sampled data may be useless.

Diversity in Node Types: VNL must support different node types, ie. Reduced-function devices (RFDs), full-function devices (FFDs) and sleeping nodes. RFDs include Low performance tiny devices, small memory sizes, low-performance processors, low bandwidth, high loss rates, etc. The software stack requirements for sensors and actuators must fit within very limited hardware configurations.

Handling of sleeping nodes is a critical requirement for VNL, as nodes might stay in sleep-mode for most of the time. Time synchronization is important for efficient forwarding of packets. Connectivity should be reliable despite unresponsive nodes due to periodic hibernation.

4.1.2 Parameterization

Policy invariants are used for parameterization (Table 3 and Table 4), eg. in order to accommodate functionality for resource constrained operations, energy-awareness & power management, QoS, diversity in node types, etc. The notation we use in the example section is 'Policy = attribute' for informal definition of the requested policy attribute value pair. As an

example, 'Policy = Low Power' represents a power saving mode, which may be instantiated differently depending on actual topology and trajectory. For an instantiated VNL the operational policy may be changed by the Modify-Invariants primitive defined in Table 3. A change to a policy may result in a change of the applied policy and virtualization scheme.

4.2 Class Descriptions

As stated above Table 1 and Table 2 provide service primitives for generic service development decoupled from aspects of operational conditions and underlying physical topology. The same primitives may be used for communication between PNs and VNs, effectively implementing common service elements covering Mobility, Multicast, Location, QoS control, Identification and security for PNs and VNs. In this context a VNL may represent any region, topology or trajectory as defined by applying the management primitives of class Region (Table 3) and class VNL (Table 4).

Class Topology (Table 5) describes the topology invariants of the class VNL in more detail. The topologies Geographic/Planar/Spanning are only example topology types and choices may be expanded.

Class Trajectory (Table 6) describes the suggested choice of trajectories for VNs, ie. Stationary or Linear or Functional. This means that the selected VN mode may be either stationary or show linear movement, or follow a spatial orbit with fixed or variable velocity component.

Migration between diverse topologies is controlled by modification of Topology-Region invariants (Topology = Geographic/Planar/Spanning/Cluster). Table 5 defines the set of topology attributes.

The functionality offered as Application Components based on the Cluster topology is described in Table 7 through Table 9.

Control of the motion of VNs is managed by setting VN-Region invariants (eg. Motion = No/Linear/Functional, as described in Table 6).

4.3 VNL Instantiation Example

This sub-section presents an example of VNL instantiation based on a road accident in an area with no cellular coverage. The cars involved in the crash autonomously establish a VN in a neighborhood of the place of the accident informing oncoming vehicles of the potential danger of roadblocks and reduced traffic capacity.

Primitives (Slots)	Parameters	Description
Create-region	(Invariants, Region-ID)	A Region with specified invariants is created
Destroy-region	(Region-ID)	Deletes specified region
Introduce-into	(Region-ID, Entity-ID)	Adds entity
Modify-invariants	(Region-ID, Entity-ID, attribute-value-pair)	Replaces invariants. This may cause side effects
Remove-from	(Region-ID, Entity-ID)	Removes entity
List-members	(Region-ID)	Identify members
Member	(Region-ID, Entity-ID)	Check for membership
Search	(Region-ID, query)	Search entities
Intersection	(list of region-IDs)	
Union	(list of region-IDs)	

Table 3 Class region and its service

Primitives (Slots)	Parameters	Description
Create-VNL	(Member-Region, Invariants, VNL-Region)	Will create a VNL from a Member-Region controlled by invariants, ie. Policy and the Attributes from Tabled 5-6. The VNL may involve single or multiple PNs or VN
Get-state	(VN-ID, State)	Returns the applications current state
Put-state	(VN-ID, State)	Resets the applications state as specified by State

Table 4 CLASS VNL and its Service

Additionally the accident is reported in both directions of the road, by collaborating vehicles, in order to reach an alarm unit coordinating such incidents.

The first alarm unit receiving the accident report replies with the required instructions for cars involved and in the accident area.

The driver in one of the cars involved in the accidents recognizes that one of his passengers has lost consciousness and needs immediate medical attention. The driver therefore sends a request for help in an area spanning 1000 metres from the point of the accident.

1 VN creation for local alerting by cars involved in the crash:

Create-region (Priority = override, Accident);
 Create-VNL (Accident, Policy = high-reliability, Topology = Geographic, Circular-Area = 2000m, Trajectory = Stationary, Accident-Area);
 Send-ID (Accident-Area, "Accident occurred in local area")

2 Alerting alarm units by cars on each side of point of accident:

Create-region (Priority = override, Accident-alert1);
 Create-region (Priority = override, Accident-alert2);
 Create-VNL (Accident-alert1, Policy = high-reliability, Trajectory = Move-North, Speed = Full, Alert-North);
 Create-VNL (Accident-alert2, Policy = high-reliability, Trajectory = Move-South, Speed = Full, Alert-South);
 Send-ID (Alert-North, Relevant data for rescue operation, Recipient = 911);
 Send-ID (Alert-South, Relevant data for rescue operation Recipient = 911);

3 Action at alarm units:

Receive-ID (Alert-North, Relevant data for rescue operation);
 Send-ID (Alert-North, Relevant rescue information);

Note that the message sent from the alarm unit applies return path routing. Much detail is left out since only the principles are focused in this presentation and since there is no space for a full in-depth description.

4 Search for local medical help

Create-region (Priority = override, Accident);
 Create-VNL (Accident, Policy = high-reliability, Topology = Cluster, Distance = 1000m, Capability = Medical, Accident-Area);

Attributes (Slots)	Value Type	Cardinality	Allowed Value
Geographic	Attribute, value pairs	Multiple	Attributes: Hop-count/ Location/ Area/
Planar	Number	Single	Diameter
Spanning	Set	Multiple	Node IDs
Cluster	Attribute, value pairs	Multiple	Attributes: Hop-count/ Distance/ Capability

Table 5 Class Topology

Attributes (Slots)	Value Type	Cardinality	Allowed Value
Stationary	Number	Multiple	WGS98 coordinates
Linear	Tuple	Single	(Direction, Velocity)
Functional	Function	Single	Location function

Table 6 Class Trajectory

Send-ID (Accident-Area, "Medical assistance needed");

A car in the neighborhood with a capable passenger receives:

Receive-ID (Accident-Area, "Medical assistance needed");

And the reply is:

Send-ID (Accident-Area, "Help available");

The car with the injured passenger picks the best alternative from the formed cluster and confirms the acceptance of the assistance. (An assistance application would have most of the steps preprogrammed.)

5 Virtual Node Layer Implementation

To implement the VNL abstraction [8], the physical nodes run emulation software that maintains a consistent view of the layer. The approach adopted is to have all the PNs in a given region run that region's VN application, locally transforming their local views of the state synchronously based on dynamic VN leadership.

The VN architecture [8] is outlined in Figure 11. The emulation of a particular VN application is divided among three main components, each of which has access to location information accurate to region granularity:

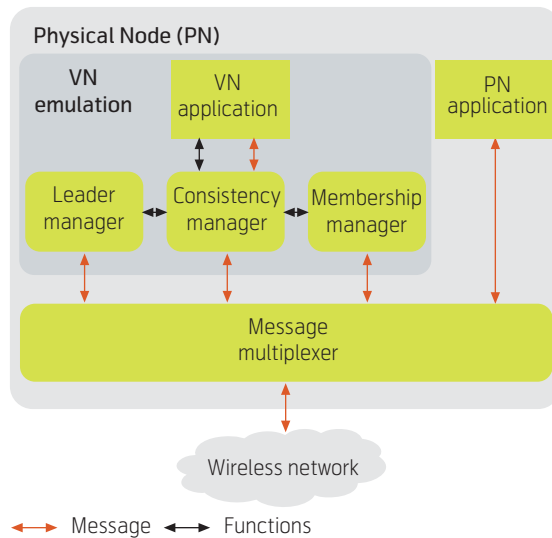


Figure 11 VN architecture

Leader Manager:

This component attempts to elect a region leader in a populated region of the network. It implements a Boolean function, called exclusively by the Consistency Manager, which returns the leader status of the physical node on which it is running.

Membership Manager:

This component retrieves the current state of the VN application being emulated in the region. It implements a function, called exclusively by the Consistency Manager, which triggers this retrieval and returns either the state or a time-out (eg. the latter may occur if there are no other nodes in the region).

Consistency Manager:

This component contains the main logic of the emulation. It is ultimately responsible for keeping the VN application state synchronized with the other physical nodes in the region. To do so, it can call upon both the leader and membership manager, and can send out its own control traffic.

Access to the VN application is mediated through the Consistency Manager, which implements the incoming and outgoing message sockets used in the application passing it messages and receiving/processing its outgoing messages.

It is required that all VN applications implement two interface functions called exclusively by the Consistency Manager: Get-state and Put-state (Table 4). The former returns the VN application's current state and the latter resets the VN application's state to the values passed as a parameter to the function. This interface allows the Consistency Manager to keep the emulated application synchronized with other nodes

in the region. For example, if a physical node enters a new region it might need to switch the application state to that of the application being emulated in the new region. The PN application(s) are unrestricted and run outside of the emulation apparatus, sending and receiving messages to and from the network directly.

5.1 Event-driven VNL

A VNL implementation [8] may be based on Reactive VNs, which are receive event-driven automata. That is, their operations are defined exclusively in terms of a MsgReceived handler which, upon being called with a received message, can transform the automata state and (potentially) return message(s) to be broadcast in response. Figure 14 provides an example. The VNs fail if and only if their region contains no physical nodes. It is assumed that messages might be lost, eg. due to collisions. The Reactive VN paradigm for the VNL was proposed in [8] as it simplifies the task of coding applications. That is, it is easier to code a message handler than it is, for example, to code an arbitrary timed I/O automaton (Appendix A).

Leader Manager: The implementation in [8] relies on a pulse-based algorithm. A leader sends out a pulse (beacon) at regular intervals. If the leader's pulse times out, the algorithm attempts to declare its PN as leader. If multiple nodes attempt to declare leadership, they elect the one with the lowest ID. If multiple nodes elect themselves leader due, perhaps, to message loss at the point of declaration they will eventually notice this situation by hearing each other's pulses. A leader who hears a pulse from another leader with a lower ID will relinquish his status.

Membership Manager: A simple join protocol asks the leader for a serialized version of the emulated VN application's state. If no leader exists, the request times out.

Consistency Manager: All physical nodes in a region locally emulate the VN application. Only the current leader, however, broadcasts on behalf of the application. For example, assume a node sends a message to the VN in a given region. All PNs in the region receive this message. The Consistency Manager running on each of these nodes passes the message to the VN application, triggering, perhaps, a state change. If the VN application reacts by broadcasting a message, the Consistency Manager checks to see if its node is the leader. If it is, it passes the broadcast messages out onto the network. Otherwise, it discards it. The Consistency Manager executes a message ordering algorithm on all received application messages, before they are passed to the local instance of the VN appli-

cation. The algorithm maintains a consistent total order on the messages across all nodes in the region to account for the possibility that a node might miss a message due to collision or temporary obstruction (possibly leading the local copy of the emulated VN application to fall out of synch with the instances on other nodes in the region). The Consistency Manager, running on the current leader, tags each outgoing VN application message with a hash of the application's state. When a non-leader node receives this message, its Consistency Manager can check that the hash tag matches the hash of its local copy of the VN application state. If the match fails, the node is out of synch, and it refreshes its state by having the Membership Manager re-perform the join protocol. Finally, the Consistency Manager is also responsible for triggering the Membership Manager to initiate the join protocol whenever the physical node has entered a new region.

6 Node and Topology Abstraction Implementation

The VN architecture shown in Figure 11 lends itself naturally to implementing the topology abstraction both as a VN application and as a PN application. Implementing topologies on top of the VNL provides the extra reliability and resilience which may be needed in certain environments, eg. for wireless sensor networks, while the topology abstractions may be run directly as PN applications in benign environments. The event-driven automata (ie. the `MsgReceived` function in Figure 14) can be running algorithms both in the physical and/or virtual space with no change. This is a major simplification which provides flexibility and opportunity for optimisation depending on the operational environment.

6.1 Node Trajectory

The Type of trajectory can be one of the following: Stationary/Linear/Function.

The Function type with a Location Function (LF) is generic covering each of the other trajectories. This implies that an implementation of the LF VN covers the implementation of all trajectory types. A special function is the 'Here' function denoting the actual current position, velocity and direction. This function follows the trajectory of a vehicle, eg. as given by a GPS device. The location function $LF(t)$ could be used in more ways:

- 1 Indicate where the CO is supposed to be at a given point in time ($LF(t)$ may differ from Here);
- 2 To control the movement of a mobile CO ($LF(t) = \text{Here}$)

$LF(t)$.Velocity denotes the speed;

$LF(t)$.Heading denotes the direction of movement.

A Region may be located by absolute coordinates or relative to Here or relative to a specific $LF(t)$.

Region.Heading = Direction relative to North or Direction relative to PN movement (ie. Forward, Reverse).

6.2 Cluster Topology

As mentioned the topology [11], [16] is a recursive organization of nodes into cells, cells into super-cells, and so on, based on an autonomous self-election of a subset of the nodes into drums, and iteratively drums self-electing to become higher level drums, and so on. The drum is also called the parent for all nodes within its cell. Figure 9 shows an example cell hierarchy. This hierarchy formation goes on iteratively until all the nodes come under one highest level cell. The self-selected drums aid in this hierarchy formation by sending periodic beacon packets. Each node can be thought of as a level 0 cell and a level 0 drum (level 0 drums do not send beacon packets). Every level k drum is at the same time also a level i drum, for all $i < k$. This hierarchy formation algorithm is distributed, with no central coordination. Drums of the same level are roughly uniformly spaced, with higher level drums more sparse than lower level drums. An address of any drum at level i is the concatenation of the address of the level $i + 1$ drum with which it associates, along with a unique identifier. This unique identifier can be any random string large enough to avoid collisions. Various sensor applications require the identifiers of nodes along with their values, and this technique leads to efficient encoding of the address of the nodes. For routing purposes [16] introduces a Distributed Hash Table (DHT) based name to address resolution scheme.

In the initial start-up period, each drum sends beacon packets (beacons) periodically, containing the beacon sequence number, the drum level, and a hop count, which aids in the hierarchy formation. These beacons are forwarded by all nodes within the hop count limit or those within the cell defined by the parent of the drum sending the beacon. In Figure 9, the beacons from node B reach all the nodes in cell level 2. The beacons sent during this initial phase also give the shortest path from any drum of level i to its associated higher level drums and the drums in the same level within its super-cell. Figure 12 shows the Cluster Formation algorithm. A drum of level i is denoted by $drum_i$, and $drum_0$ denotes the nodes. Beacons sent by $drum_i$ are denoted by $beacon_i$. D1 is the fundamental cell (ie. cell level 1) diameter and is dependent of the cluster size required by the application.

The value of α is the ratio between the beacon hop limits for consecutive drum levels.

The drums (also level 0 drums) wait a random time between 0 and T_{max} , before deciding whether to become a higher level drum or associate with another drum. The association scope of the drums, determined by the hop limit of the beacons, increase geometrically with the level of drum. Lines 8-10 of the algorithm ensure that the beacons of any drum reach all nodes within its super-cell (enhances routing scalability). Lines 11-13 ensure that no drums of same level form too close to each other, as that reduces the efficiency of the clustering. Lines 14-16 make the hierarchy evolve such that the drums are always asso-

ciated with closest higher level drum. The number of levels formed in the hierarchy is of $O(\log(N))$, where N is the number of nodes in the network. As a simple analysis of the cost of the algorithm, if instantaneous propagation is assumed, then all level i drums are separated by D_i hops, with very high probability, and so the total startup phase is of $O(T_{max} \log(N))$.

Application specified filters, called Selectors, are introduced [11] to align this hierarchy to match the collaboration and communication sets of nodes. A Selector is a tuple of {attribute; value; operator}, where attribute is any application specified variable, and value is a valid element from the range of the attribute. The operator is a binary operation (such as $>$, $<$, or $=$) with value being one of the operands. The definition of a Selector is extended to form: Selectors = Selectors?Selectors | Selectors?Selectors | Selectors | Selector | null. The values for the attributes are assumed to be shared between the application, sensor, and the networking layer at a node, which enables the Selectors to be evaluated at the networking layer.

An operator needing a time-series of previous attribute values might entail the sharing of whole data structures of application computed values. The beacons of drums are forwarded by a node if the hop count in the beacon packet is less than a specific value and may also be made conditional to the specified Selectors. For empty Selectors, the effect is to forward the beacons based only on hop count.

Re-clustering of the network hierarchy to adapt it closely to the communication flow is initiated by the application locally in cells where adaptation is needed. This is best judged by the application, as the networking layer does not have any knowledge of the application logic or how the sensed values influence the communication. Figure 13 shows the Re-cluster algorithm. The parameters for cluster formation are changed locally to reflect current communication patterns. The Selectors encode the criterion for the new cluster formation in the Split Phase of the algorithm. After re-clustering, some nodes might become orphans (nodes without parent) or some cells might be smaller than optimal. These nodes or cells then merge with neighbouring cells meeting the criterion in the Merge Phase of the algorithm. The rest of the clusters in different areas are not changed. Hence, this re-clustering takes place locally only where necessary and invokes no long range messaging, ie. it is bound by the hop count.

6.2.1 Efficient Communications in Cluster Topologies

The algorithms in Figure 12 and Figure 13 are used by the applications to form a clustered hierarchy and

```

REPEAT
  Drumi wait for a random time up to  $T_{max}$ 
  IF Drumi does not hear a higher level beacon, and is not the highest level
  THEN
    Step up to leveli+1 and start sending periodic beacons with hop limit of
     $D_{i+1} = \alpha \times D_i$ 
  ELSE
    Associates with the nearest Drumi+1.
  FI
8  IF Drumi hears any non-duplicate beacon by a drum in its super-cell
  THEN
    Drumi forwards the beacon.
10 FI
11 IF Drumi hears any beaconi <  $D_i$  hops away THEN
    The drum with the lower id steps down to leveli-1
13 FI
14 IF Drumi hears any beaconi+1, which is closer than current beaconi
  THEN
    Drumi associates itself with the closer drum; % Not reported back to the
    Drum;
16 FI
UNTIL All nodes are assigned stable addresses
Topology-ID:= Highest level Drum.ID;

```

Figure 12 Initial cluster formation

```

Drum sends out beacons with Selectors
Nodes matching the Selectors (re)select the Drum as their parent
IF Any node become orphan or cell is sub-optimal THEN
  It sends Solicit Beacons with specified Selectors
  Nodes matching the Selectors respond with Beacons
  Choose the Drum most suitable with respect to the Selectors, and
  become a child
FI;
Topology-ID:= Drum.ID;

```

Figure 13 Re-cluster (Selectors)

to adapt it for efficient implementation. There are different mechanisms for efficient support of the different primitives. The basic assumption is the existence of bidirectional wireless links, which is true for most commonly used wireless MAC protocols. The following gives a summary related to the target for communications.

With the parent node: The drum beacons that are used to form the cluster hierarchy is utilized to route from and to parent node, by following the path or reverse path of the beacons respectively. If the path breaks, due to nodes in the path moving away or dying, then local route repair is done to find a new route. The drum whose path breaks, sends out beacons for a short interval to repair the broken path.

With the peer nodes: At any level, the peer nodes need to be able to communicate with each other efficiently. This is achieved by expanding the scope of the beacons for the drums. The beacon packet of a level n drum is also forwarded by all nodes in the level $n + 1$ cell of the originating drum. The reverse path is followed to reach each peer. The configuration is only done in the initial start-up period or when a path breakage is detected. Multicasting at network or MAC layer (if possible) can be done to prevent duplicate packets along common part of the paths. For example in Figure 9, beacon packets from node A is flooded to the whole Cell Level 2. And hence B, C and D know of the shortest path to A. The above technique leads to minimum latency communication from any node to the rest of its peers, using shortest path. But, it also may lead to higher cost in terms of number of forwards. Alternatively a Minimum Spanning Tree can be formed between the peer nodes. This leads to lesser number of forwards, but also leads to higher maximum latency.

Within the cell: Communication from any node to the whole cell can be achieved by simple flooding within the scope of the cell, whereby each node forwards a packet exactly once. However, this is very sub-optimal and leads to many redundant broadcasts especially in a dense network. A strategy for optimal cell flooding is described in [11]. In a NBMA (Non Broadcast Multiple Access) network the traffic could efficiently be routed via a spanning tree rooted at the drum of the cell.

Node to Node: To route a data packet, the packet is forwarded according to the hierarchical address of the packet's destination, routing recursively at each level towards the drum for the destination node's cell. To route towards a drum at a given level, packets are routed following the reverse path of the most recent beacon received from that drum. Once the packet

```

DO Forever
MsgReceived (Message-Type, Payload);
CASE Message-Type IN:

    Create-region (Invariants, Region-ID)
    Destroy-region (Region-ID)
    Introduce-into (Region-ID, Entity-ID)
    Modify-invariants (Region-ID, Entity-ID, attribute-value-pair)
    Remove-from (Region-ID, Entity-ID)
    List-members (Region-ID)
    Member (Region-ID, Entity-ID)
    Search (Region-ID, query)

    Create-VNL (Member-Region, Invariants, VNL-Region)

Get-state (VN-ID, State):    % Returns the VN application's current state.
Put-state (VN-ID, State):    % Resets the VN application's state.
Receive-ID (OID, Data):      % Incoming Data received by the VN.
IF OID = MYID THEN "Process incoming Data" ELSE Relay (OID, Data) FI;
ESAC;
OD Forever;

```

Figure 14 Event-driven automata framework

reaches the fundamental cell of the destination, any effective traditional ad hoc network routing protocol can be used, since the size of a fundamental cell is limited. The architectural framework and algorithms for efficient and scaleable node to node communications and routing in dynamic ad hoc networks are well described in [16].

6.3 Event-driven Automata Framework

The event-driven automata framework is described in Figure 14. The routing and forwarding in the hierarchy of cells are described in [16].

6.4 Application Components

The cell hierarchy opens the opportunity for efficient implementation of the following functionality as part of the application components layer as shown in Figure 5.

6.4.1 Capability Discovery

Discovery primitives [11] can be invoked by any node to give itself information about related nodes. In the architecture presented here these primitives are classified to be part of Management. This information is gathered periodically, with a period as specified by the application, and the application is informed of any changes in the information. This procedure is continuous, either triggered by node failures or additions. The information might contain the identifiers of the set of nodes, their locations, link quality and number of hops to each of them, and resource (eg. remaining battery or available sensors) present in each of them.

Primitives	Parameters	Description
Parent Information	(Level)	Returns information about the parent of the node at the specified level. When invoked on node 6 with level 1 returns information about node B, and with level 2 returns information about node C.
Peer Information	(Level)	Returns information about the peers of the node at the specified level. When invoked on node B at level 1, returns information on the set of nodes A, B, C and D.
Cell Information	(Level)	When invoked on node A, returns information on the set of nodes 1, 2, 3, 4, 5 and A.

Table 7 Class Capability discovery service Primitives (Slots)

The information learned from these discovery primitives (Table 7) can be used to configure the Selectors with any specified criterion. For example, the Selectors might be specified to filter nodes within a specified geographical distance or with higher than a specific link quality. There are three primitives proposed [11]. As a node (ie. drum) can be simultaneous in different levels, a Level is specified for each primitive to specify the level for which the information is required. The format for representing the gathered information is implementation dependent. The description in Table 7 refers to Figure 9.

6.4.2 Put and Get Primitives

In the architecture [11] both kinds of communication models Put and Get are supported. In the architecture presented here these primitives are classified to be part of Application components (Figure 5). With Put, a node sends data to its cell, parent, or peers, whereas in Get, a node solicits data from its cell, parent, or peers. The Put primitives correspond to the push paradigm, and the Get primitives correspond to the pull paradigm. Different applications might be opti-

mized using different paradigms. The Put primitive can be implemented in multiple ways: stored locally, sent immediately to the designated scope, or cached at different intermediate locations. Similarly, the Get Primitive implementation might involve either fetching remote data or local retrieval. The specific implementation depends on the application characteristics, and using management functionality for configuring the best alternative is an attractive approach.

The description in Table 8 refers to Figure 9. For node A, the parent is node C; the peers at level 1 are B, C and D; and the peers at level 0 as well as the cell at level 1 (ie. the fundamental cell) are nodes 1, 2, 3, 4, and 5.

6.4.3 Data Fusion Primitives

The proposal [11] also supports Reduction primitives that use an associative operator (such as sum, max, or min) to reduce an attribute across all the nodes in a specified region. These reduction primitives can be implemented using Get and Put, but efficient implementations can take advantage of local reductions

Primitives	Parameters	Description
PutParent	(Attribute, Value)	Attribute is sent to the parent node. (When called on Node A, the data is sent to node C)
PutCell	(Level, Selectors, Attribute, Value)	Level can be at most one level higher than the node using this primitive. So, a node of level0 can send message to the fundamental cell. In general, a drumi can send message to all nodes in the level+1 cell. For Node A, PutCell called with level 1 delivers data to nodes 1, 2, 3, 4 and 5, while called with level 2 delivers data to all the nodes marked by Cell level 2. The targeted nodes can filter the receipt of the Data using the Selectors.
PutPeer	(Level, Selectors, Attribute, Value)	The level can be at most same as the level of the node using the interface. This interface provides the same functionality as PutCell for level0 nodes. For node A, level 1 delivers the data to nodes B, C and D (Peer Level 1)
GetParent	(Attribute)	The value of the attribute is solicited from the parent node
GetCell	(Level, Selectors, Attribute)	Level can be at most one level higher than the node using this interface. In this interfaces, Data is received from the cell nodes matching the Selectors
GetPeer	(Level, Selectors, Attribute)	The level can be at most same as the level of the node using the interface. This interface provides same functionality as GetCell for level0 nodes

Table 8 Class Put and Get Service Primitives (Slots)

Primitives	Parameters	Description
ReduceCell	(Level, Selectors, Attribute, Operator)	This interface is applied on the attribute for all nodes in the cell specified by level and Selectors. And the reduced attribute value is stored locally. For example for operator max, the maximum attribute value within the cell is returned by this interface
ReducePeer	(Level, Selectors, Attribute, Operator)	Similar interface where the scope is all the peer nodes at the specified level

Table 9 Class Fusion Service Primitives (Slots)

while propagating the values. Reduction primitives are invoked either on cells or peers.

7 Conclusion

The VNL management service ontology extends the abstraction power of regions by allowing applications development to take place without considering the characteristics of the underlying network topology and application level connectivity. Management primitives are defined to adopt the most efficient virtualization depending on the environment of deployment. The same service implementation will thereby be adaptable to its current environment.

An instantiation of the VNL can be made by the application through the API, or remotely by sending the VN ontology to other nodes forming an abstract region. This represents the concept of data driven (ie. ontology driven) virtual mobile agents.

The VNL architecture is well suited not only for implementing the VN concept of collaborating Physical Nodes (PN), but it lends itself naturally both to implementing the topology abstraction as a VN application and as a PN application. Implementing topologies on top of the VNL provides the extra reliability and resilience which may be needed in certain environments, eg. for wireless sensor networks, while the topology abstractions may be run directly as PN applications in benign environments. Applications can be operating both in the physical and/or virtual space running identical code (eg. in the form of event-driven automata). This is a major simplification which provides flexibility and opportunity for optimisation depending on the operational environment.

References

- 1 Grønbaek, I, Jakobsson, S. *High level architecture for support of CO services*. Fornebu, Telenor R&I, 2007. (Research Report R37/2007)
- 2 Grønbaek, I et al. *Abstract Service API for COs*. Fornebu, Telenor R&I, 2007. (Research Report R18/2007)
- 3 Grønbaek, I. *Connecting objects in the Internet of Things (IoT)*. Fornebu, Telenor R&I, 2008. (Research Report R6/2008)
- 4 IETF. *Host Identity Protocol (HIP) Architecture*. RFC 4423, May 2006.
- 5 Audestad, J A, Grønbaek, I, Svaet, S. *Connected Objects Platform Specification*. Fornebu, Telenor R&I, 2009. (Research Report R 2/2009)
- 6 Raman, B, Chebrolu, K. Sensor networks : A Critique Of 'Sensor Networks' from a Systems Perspective. *ACM SIGCOMM Computer Communication Review*, 38 (3), 2008.
- 7 *Representational State Transfer*. August 18, 2009 - URL: http://en.wikipedia.org/wiki/Representational_State_Transfer.
- 8 Brown, M et al. The virtual node layer : a programming abstraction for wireless sensor networks. *ACM SIGBED Review archive*, 4 (3), 2007. (Special issue on the workshop on wireless sensor network architecture, April-2007) New York, NY, USA, ACM.
- 9 Dolev, S et al. Virtual Mobile Nodes for Mobile Ad Hoc Networks. *International Conference on Principles of Distributed Computing (DISC)*, 2004.
- 10 Welsh, M, Mainland, G. Programming sensor networks using abstract regions. **In:** *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)*, March 2004.
- 11 Pal Chaudhuri, S et al. Design of Adaptive Overlays for Multi-scale Communication in Sensor Networks. *Proc. Int. Conf. on Distributed Computing in Sensor Systems*, LNCS, Springer 2005.
- 12 Dolev, S et al. Self-Stabilizing Mobile Node Location Management and Message Routing.

Seventh International Symposium on Self-Stabilizing Systems (SSS 2005), Barcelona, Spain, October 26-27, 2005. Also, Technical Report MIT-LCS-TR-999, MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, August 2005.

- 13 Nolte, T, Lynch, N A. Self-stabilization and Virtual Node Layer Emulations. T Masuzawa, S Tixeuil (eds.) *Stabilization, Safety, and Security of Distributed Systems, 9th International Symposium, SSS 2007*, Paris, France, November 14-16, 2007, 394-408.
- 14 Sollins, K R. Designing for Scale and Differentiation. In: *Proc. ACM SIGCOMM 2003 Workshop on Future Directions in Network*, Karlsruhe, Germany, August 2003.
- 15 Grønbaek, I, Biswas, P. Ontology-based Abstractions for M2M Virtual Nodes and Topologies. *ICUMT 2009*, St.Petersburg, October 2009.
- 16 Du, S et al. Safari : A self-organizing, hierarchical architecture for scalable ad hoc networking. *Ad Hoc Networks*, 6, 485-507. Elsevier, 2008.
- 17 Garland, S J et al. *TIOA Tutorial*. Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, May 22, 2005.
- 18 Kaynar, D et al. *The Theory of Timed I/O Automata*. Morgan and Claypool Publishers, 2006.

Appendix A – TIOA Introduction

This introduction is an excerpt from [17]. A formal treatment of the theory can be found in [18]. Timed input/output automata provide a mathematical model suitable for describing time-dependent behaviour in concurrent systems. The model provides a precise way of describing and reasoning about system components that interact with each other through discrete actions as well as the continuous evolution of internal state components over time.

The fundamental object in the TIOA framework is a timed (I/O) automaton, which is a kind of nondeterministic, possibly infinite-state, state machine. The state of a timed automaton is described by a valuation of state variables that are internal to the automaton. The state of a timed automaton can change in two ways: instantaneously, by the occurrence of a discrete transition, or over an interval of time via a trajectory, which is a function that describes the evolution of the state variables. Trajectories may be continuous or discontinuous functions.

TIOA transitions are associated with named actions, which are classified as input, output, or internal. Input and output actions are used for communication with the automaton's environment, whereas internal actions are visible only to the automaton itself. The input actions are assumed not to be under the automaton's control, whereas the automaton itself controls which output and internal actions should be performed.

The communication of a timed automaton with its environment is limited to discrete transitions associated with actions shared between the automaton and its environment. The TIOA framework does not model continuous information flow between a timed automaton and its environment; this kind of modelling would require a full-fledged hybrid automaton model.

The time domain in TIOA is the set of real numbers. States of automata consist of valuations of variables. Each variable has both a static type, which defines the set of values it may assume, and a dynamic type, which gives the set of trajectories it may follow. We assume that dynamic types are closed under some simple operations: shifting the time domain, taking subintervals, and pasting together intervals.

A trajectory for a set V of variables describes the evolution of the variables in V over time; formally, it is a function from a time interval that starts with 0 to valuations of V ; that is, a trajectory defines a value for each variable at each time in the interval. A typical example of a timed automaton is a time-bounded FIFO message channel. Formally, a timed (I/O) automaton A consists of the following six components:

- A set X of internal variables;
- A set Q , which is a subset of all possible valuations of X ;
- A set of initial states, which is a non-empty subset of the set of all states;
- A signature, which lists disjoint sets of input, output, and internal actions of A ;
- A discrete transition relation, which contains triples of the form (state, action, state); and
- A set of trajectories for X such that $\tau(t) \in Q$ for every $\tau \in T$ and every t in the domain of τ .

An action π is said to be enabled in a state s if there is another state s' such that (s, π, s') is a transition of the automaton. Input actions are enabled in every

state; ie. automata are not able to ‘block’ input actions from occurring. The external actions of an automaton consist of its input and output actions. The transition relation is usually described in precondition-effect style, which groups together all transitions that involve a particular type of action into a single piece of code. The precondition is a predicate (that is, a boolean-valued expression) on the state indicating the conditions under which the action is permitted to occur. The effect describes the changes that occur as a result of the action, either in the form of a simple program or in the form of a predicate relating the pre-state and the post-state (ie. the states before and after the action occurs). Actions are executed indivisibly. Trajectories are defined using invariants, algebraic and differential equations, and ‘urgency’ conditions that specify when time must stop to allow a discrete action to occur.

Using TIOA to Formalize Descriptions of Timed I/O Automata

We illustrate the nature of timed automata, as well as the use of TIOA to define the automata, by a few simple examples. Figure A.1 contains a simple TIOA description for an automaton, Timeout ($u:\text{Real}$, $M:\text{Type}$), that awaits the receipt of a message of type M from another process. If u time units elapse without such a message arriving, the automaton performs a timeout action, thereby ‘suspecting’ the other process. When a message arrives, it ‘unsuspects’ the other process. The automaton may suspect and unsuspect repeatedly. The automaton is parameterized by the timeout period u and the type M of the messages received by Timeout ($u:\text{Real}$, $M:\text{Type}$). The automaton Timeout has two state variables: `suspected` is a boolean that is set to true when a timeout occurs, and `clock` is a real number that represents a timer running at the same speed as real-time. The initial value of `suspected` and `clock` are false and 0. The value of the automaton parameter u is constrained to be strictly greater than 0. The transitions of the automaton Timeout are given in precondition/effect style. The input action `receive(m)` has no precondition, which is equivalent to having true as its precondition. This is the case for all input actions; that is, every input action in every automaton is enabled in every state. The effect of `receive` is to reset `clock` to 0 and to set `suspected` to false (in case it had been true before). The output action `timeout` can occur only when it is enabled, that is, only in states in which `suspected` is false and `clock = u`. Its effect is to set `suspected` to true. The two trajectory definitions `suspected` and `notsuspected` correspond to two ‘modes’ of the Timeout automaton. While the `suspected` flag is false, the clock advances with rate 1, that is, with the same rate as real-time, and time cannot go beyond the point at which `clock = u`. While the `suspected` flag is true there is no condition on time-passage; the clock may keep advancing with rate 1. Note that trajectories

do not need to be followed until a stopping condition is reached; however, if a stopping condition is reached then time must stop. At this point, a discrete action may occur if it is enabled.

Trajectories of an automaton are defined following the keyword *trajectories*. A trajectory definition consists of a name, an invariant, an *evolve* clause, and a stopping condition. More than one trajectory definition can be used to define trajectories of an automaton. For example, the automaton Timeout in Figure A.1 has two trajectory definitions.

Each trajectory definition defines a set of trajectories; the set of all trajectories for an automaton is the concatenation closure of all of these sets (see [3] for the definition of concatenation for trajectories). A trajectory belongs to the set of trajectories defined by a trajectory definition if it satisfies the predicate in its *invariant* clause, the differential equations in the *evolve* clause and the stopping condition expressed by the *stop when* clause. The stopping condition is satisfied by a trajectory if the only state in which the condition holds is the last state of that trajectory. In other words, time cannot advance once the stopping condition becomes true.

Automaton Timeout($u:\text{Real}$, $M:\text{type}$) where $u > 0$

signature

input receive($m:M$)

output timeout

states

suspected: Bool:= false,

clock Real:= 0

transitions

input receive(m)

eff clock:= 0;

suspected:= false

output timeout

pre not suspected \wedge clock= u

eff suspected:= true

trajectories

trajdef suspected

invariant suspected

evolve d(clock)= 1

trajdef notsuspected

invariant not suspected

stop when clock= u

evolve d(clock)= 1

Legend: eff signifies effect of action

Figure A.1 TIOA description of a timeout process

Inge Grønþæk received his M.Sc. degree in computer science from the Technical University, Trondheim, Norway, in 1977. He has since been involved in protocol design and implementation for systems involving circuit switching, packet switching, and message handling. He was for a period seconded from Siemens Norway A/S to NATO at SHAPE Technical Centre (STC) in the Hague, Holland, where he was working in the area of protocol standardisation. In 1994 he took a position as Chief of technical development at the Networks division of the Norwegian Telecom. In September 1998 he transferred to Telenor (former Norwegian Telecom) R&D as Senior Adviser. In the period 1999 to 2002 he was the Telenor representative in the international 3G.IP Focus Group aiming at applying IP technology in third generation mobile systems. His current focus is in the area of IP based network evolution, in particular on architecture and service production for M2M and connected objects. He is currently representing Telenor in the ETSI TC M2M.

inge.gronbak@telenor.com