Wide-Area SMC Interaction, Implementation and Emulation

Stephen Strowes*, Naranker Dulay†, Steven Heeps*, Sye Loong Keoh†, Emil Lupu†,
Alberto Egon Schaeffer-Filho†, Morris Sloman† and Joe Sventek*
*Department of Computing Science, University of Glasgow
{sds,heeps,joe}@dcs.gla.ac.uk
†Department of Computing, Imperial College London
{n.dulay,slk,e.c.lupu,aschaeff,m.sloman}@doc.ic.ac.uk

Abstract— **The primary components in a Self Managed Cell (SMC) – the event bus, the policy management service, and the discovery service – are required regardless of the scale of the SMC. However, the behaviour of core services may necessarily be altered to suit the environment within which an SMC operates. This paper discusses the design of core services (primarily, the event bus and discovery service) in wide-area SMCs. Delay-tolerant networking between SMCs is also discussed, as is the implementation of core services leading to an emulated network of SMCs. As the basis for a "healthmap" capable of representing patient data across a geographic region, the discussion on wide-area SMCs leads into cursory discussion of geographical imaging and visualisation systems.**

I. Introduction

The second demonstrator specified by the AMUSe project is a "health map", intended to demonstrate the scalability and flexibility of the Self Managed Cell (SMC) architecture [1]. With this as our goal, we defined the asthma scenario (whereby a patient's asthma inhaler usage is logged at the patient and carried in a delay-tolerant manner, ultimately to be stored at the GP surgery), and defined the behaviour of core services at larger scales [2].

The AMUSe project defines the SMC as a repeating pattern of management components and multiple managed nodes intended to manage local-area SMCs and wide-area SMCs alike. A managed node can be remote, dumb sensor, a remote "intelligent" devices (including other SMCs), or software services. The required core services are consistent across all SMCs regardless of environment and scale. All SMCs require a policy service as the centralised management component, an event bus for the conveyance of management traffic between components, and a discovery service to handle the process of locating other SMCs and group members.

The environments in which we expect these core services to run, however, can vary widely and so the behaviour of the services must be altered to suit.

To contrast the mobile personal-area SMC we have previously focussed on, the wide-area SMCs we envision in [2] will be required to run on stationary IP-based local-area and wide-area networks. The notable difference at larger scales is the behaviour of the discovery service, which varies more considerably than the behaviour of the event bus at differing scales.

In a wide-area SMC we are more able to assume permanent network components which do not appear or disappear frequently (e.g., hosts, printers, databases, etc), which are often connected via a wired network infrastructure. We must also consider that this network environment potentially allows for many more networked components falling under the jurisdiction of one SMC than the localised ZigBee-based personal-area SMC, owing mainly to the physical characteristics of the different types of networks.

This paper reviews the design considerations and implementation experience derived from building a proof-of-concept and proof-of-existence wide-area SMC and an emulation of a real-life set of interacting SMCs. We also consider interactions between mobile and stationary SMCs, and how to transfer data between SMCs using the principles of delay-tolerant networking.

While the construction of an actual healthmap was not achieved due to time constraints, the central issue of core service scalability has been considered in depth. We believe the remainder of the healthmap work to pose no major problems and to be a matter of finding time and resources to build an implementation.

The remainder of this paper is structured as follows: Section II discusses the desired behaviour of the core services within a wide-area SMC. Section III briefly covers the concept of linking together mobile and stationary SMCs, while Section IV discusses delay-tolerant networking and how it applies to our networked SMCs. Section V covers implementation details, and Section VI the actual experimental setup. The paper is concluded over Sections VII, VIII and IX which cover data visualisation for the healthmap, current status, and the conclusion respectively.

II. Core Services in the Wide Area

This section discusses each of the core services separately, and the requirements placed on each of those in wide-area environments. The rationale supporting the design of the wide-area SMC design is detailed more thoroughly in [2].

A. Event Bus

The event bus is required to route management events from services or devices which are members of an SMC onto any interested parties within the SMC which have subscribed to receive events matching certain criteria.

It is essential that the communication of management events satisfy *at most once* semantics – i.e. all events are delivered to each interested component exactly once if the interested component is still a member of the SMC.

Since there may be causal relationships between pairs of events from the same source, the event bus must also guarantee that all events from a particular sender are delivered to each interested receiver in the order sent. Note that this does not say anything about delivery order between events from *different* sending components, as this would require a model of causality for the entire SMC.

One additional caveat if we are to consider a wide-area event bus is scalability in terms of potential throughput; while we do not expect management traffic in a sensibly constructed SMC to be onerous, we should design to scale as far as is reasonable. (i.e., where there is not an equivalent performance penalty hit in doing so.)

The policy service generates management traffic, but it does not do so exclusively; for example, the discovery service uses the event bus to carry membership announcements, which are certainly also management traffic. Application traffic may be carried over the event bus, but applications are free to choose to use whichever communication mechanisms are available to them, of which the event bus is only one.

Siena is a content-based publish/subscribe event service, which can be used in a centralised manner with clients connecting to a central server [3]. Siena also provides scalability by allowing these servers to connect to each other in a hierarchical fashion, with events then routed over the hierarchy. The hierarchical structure is fixed, and so there is no scope for reordering of events if routing tables at Siena nodes change. Siena can also use TCP links between nodes, so events in transit should not be reordered.

The compact event bus designed for smaller wireless environments was designed to act as a centralised event forwarding mechanism based on the Siena algorithms [4], and so utilising Siena for the wide-area SMC allows for easy integration of the two, without the need to translate events between two different (and potentially complex) event description languages.

B. Discovery Service

Each SMC requires a discovery service, which implements a protocol to initiate communications between the SMC and a new device or service, and to subsequently grant or deny group membership and integration of that device into the cell. On granting a new device membership to the SMC, the discovery service will fire a *newMember* event describing that member (or a *newSMC* event if the new member is another SMC). There also exist symmetric *purgeMember* and *purgeSMC* events.

The discovery service has several purposes constituting management of group membership: to handle admission of new nodes into the cell (employing authentication specific to the application); to handle handle the removal of nodes which have left the cell (either cleanly by announcing their departure, or through being physically removed, disconnected, etc); and to maintain connectivity to nodes while they are part of the cell. Services within an SMC should be able to use the discovery service to lookup other services.

The protocol should mask transient disconnections between components, e.g. a nurse leaving a room for a short while to tend to another patient before coming back, or a wireless connection to a desktop PC being temporarily disrupted.

The discovery service for the personal-area SMC is designed for heavily localised, connectionless, wireless environments which make the broadcast-based nature of this discovery protocol appropriate. In a more traditional networking environment, periodic broadcasts in this manner are wasteful.

In more traditional networks the behaviour of the software components must be reversed, such that devices and services don't listen for the discovery service but instead locate the discovery service (possibly by one of a variety of mechanisms: DNS, multicast, pre-loaded addresses to ping for the existence of the discovery service), then initiate communication. Thus, the actual process of locating nodes is altered to suit the environment.

Note that we can view two distinct phases to any discovery protocol: initial discovery of services and other SMCs, followed by subsequent restriction of that set of discovered items (for example: locate a printer, then choose the most appropriate; wireless sensors must locate the correct patient's discovery service and choose the correct one). The latter phase can be solved by either of the following methods, for example: the discovery service could restrict the viewable set of components to a given node, provided some context (e.g., a PC configured to identify itself as belonging to a particular room may choose the printer also in that room); or a "two-button" mechanism to enable devices to identify each other amongst others (thus solving the problem faced by the wireless sensor).

The latter phase here can be deemed as an application or environment specific characteristic of SMC interactions. We focus largely on the initial phase. (Although, in both the personal-area SMC and wide-area SMC we can offer some scope for service selection: query based with pre-configured parameters in the printer example, and proximity based for wireless sensors.)

1) Discovery Services: Service discovery in traditional networking environments utilises directory-based mechanisms to locate service providers. Access to these directories is provided by protocols such as the lightweight directory-access protocol (LDAP).

LDAP is a thoroughly documented protocol [5] which provides access to various directory services (e.g., an X.500 directory). Hosts can bind or unbind from the LDAP server, and add, remove or modify entries while they are connected. We could use LDAP to advertise hosts and services. JNDI is an interface to allow Java components to access various standard directory services such as LDAP [6], which may prove useful for interfacing our existing Java components with traditional network services to offer a discovery service.

Service discovery protocols generally rely on the existence of a directory service to arrive at what we would consider a discovery service. These directories may be centralised or distributed. Some existing service discovery protocols include Zeroconf, SLP, UPnP, and Jini.

Zeroconf aims to offer easy setup for IP networks. Various implementations exist, such as Apple's "Bonjour", and Linux's "Avahi". Zeroconf is based on mDNS, multicast-DNS, whereby end hosts store their own list of DNS records describing services they offer. Lookups are performed by multicasting to a known address and awaiting responses from matching hosts. This suggests that Zeroconf cannot scale with the network, since each host receives all lookup packets sent over the network.

The Service Location Protocol (SLP) is defined by the IETF for use over IP based network environments [7]. Directory agents (DAs) in an SLP environment hold the services registered by service agents (SAs) for other entities to use; an SA attempts to locate a DA by multicasting to a known address on joining the network, and DAs periodically multicast a heartbeat packet to inform the rest of the network of its presence. The protocol can work in the presence of a DA or without. If no DA is present, nodes multicast service requests and SAs respond directly. Multiple DAs can connect to each other in order to allow this setup to scale. SLP is supported by various forms of Linux and MacOSX, and open C and Java implementations are available for download [8]. Its use of IP multicast may make SLP unsuitable in some network situations.

Universal Plug and Play (UPnP) is the output of the UPnP Forum and uses a discovery protocol based on the Simple Service Discovery Protocol (SSDP) [9]. UPnP uses non-standard HTTP over UDP across either multicast addresses or unicast links. Services announce their presence by sending an ANNOUNCE message, and can query the network for a resource with an OPTIONS message [10]. Beyond discovery, the UPnP protocol tackles additional functionality, such as control & management, event notification, NAT traversal, amongst others. UPnP is a complex piece of software aiming to solve a larger problem set than just service discovery.

JINI is an open network architecture for the construction of distributed systems. JINI services locate the directory service by requiring that new services or devices multicast a *presence announcement* to a known address. On receiving this announcement, the directory service communicates with the new device using remote method invocation (RMI), passing objects to the new device to allow it to *join* with the directory service, and to perform *lookup* operations. This requirement forces application to use Java's RMI, which may not suit all SMC settings.

Our requirements suggest that we need little more than a directory service, perhaps with some additional code to provide exactly the semantics we require; sufficient generality such as to not restrict the SMC platform to a specific programming language or operating system suggest that either SLP or an LDAP server are appropriate for our purposes.

For the implementation of these services, we chose SLP due to the simplicity of the multicast mechanism on which it is based and the availability of source code.

C. *Policy Service*

Policies provide the means of specifying the adaptation strategy for autonomic management [11]. There are two distinct types of policy: authorisation policies, which specify what resources the components assigned to a role can access, and obligation policies (event-condition-action rules) which specify how components/services react to events and interact with other components/services.

When a device is discovered and granted membership of an SMC the appropriate policies, based on device type, are deployed to it. This is triggered by a *newMember* event generated by the discovery service. Policies can be added, removed, enabled and disabled to change the behaviour of cell components at runtime. Policies also govern the behaviour of the discovery service and the policy service itself, enabling these to be tailored to specific situations.

One key advantage offered by the policy service is the ability to load 'missions' from one SMC onto another; a mission is a task which that SMC will carry out even while disconnected from the parent or peer SMC which loaded the mission. Thus, we are capable of automatically configuring an SMC to perform a task (e.g., take readings over a period of time) while it is mobile.

III. Bridging the Gap

Bridging the gap between the mobile environment, with ZigBee-based SMCs, and stationary SMCs on an Ethernet network would simply require a gateway to transfer/translate messages between the two environments. Given the differing behaviours of the two realms, message translation may be an important aspect of the gateway's behaviour.

Using Figure 1 as a reference, the gateway uses one or more ZigBee transceivers to periodically broadcast the SMC ID of the stationary network to which it is attached; this SMC could be a hospital ward, or a hospital building for example. To accommodate ZigBee-based SMCs, the gateway performs membership admittance as per a mobile discovery service and then advertises the mobile SMC in the SLP DA. It monitors continued group membership via received unicasts from mobile SMCs.

The gateway translates messages between formats suitable for the larger-scale and smaller-scale services, as appropriate.

A gateway can also become an SLP DA, thus directly handling the membership of mobile SMCs and visibility of mobile SMCs to the rest of the wide-area SMC. This DA can share the responsibility of the discovery service with another agent located in the PC marked "SMC Core". Likewise, the gateway can join a distributed Siena system to help with event handling to and from mobile entities.

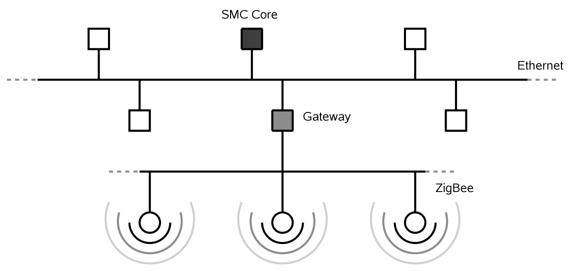

Fig. 1. Bridge between traditional Ethernet network and ZigBee network, over which SMC services will be able to communicate.

IV. Delay-Tolerant Networking

Given our mix of mobile components based on battery-powered wireless technologies and mains-powered wired components, we can obviously draw borders between a fixed network infrastructure and mobile networked regions, with transient links connecting the two. It is apparent then that mechanisms may be required to ease the data transfer between the two different networks.

Delay-tolerant networking (DTN) suits our SMC architecture given the mobile nature of (some) cells within most potential environments. A DTN defines regions which nodes inhabit; inter-region routing takes place between nodes on the border of two regions, and intra-region routing is attempted once a “bundle” of data has arrived at its target region, [12]. DTN region labelling is application-specific, but it is easy to imagine two extremes: one, where each SMC is a region, and the other where all SMCs inhabit one large shared region, and varying levels of fragmentation in-between.

Delay-tolerant networking within the context of interacting SMCs provides some nice, if subtle, behaviour: the DTN layer can react to the very same management events that the rest of an SMC reacts to. For example, the DTN layer does not need to probe for new connections. Instead, it will accept a *newSMC* event and react to it – if the *newSMC* belongs to a region which the DTN layer is configured to connect to, then a connection can be established, triggering the transmission of previously stored data in direct response to this *newSMC* event. Likewise, *purgeSMC* events close connections, clear up state, and ensure that undelivered data is retained for later delivery.

The demo application discussed in Section VI sends data via the DTN, and forgets about that data once acknowledged by the next node. The DTN implementation uses what is referred to as custody transfer [12] for data transfer; data is passed to a custodian who acknowledges receipt, who then forwards to the next potential custodian or final destination. Custody transfers fit this scenario if we choose to place trust in the NHS nodes; other scenarios with untrusted custodians may require different behaviour of the DTN layer.

Data must be stored on non-volatile storage until the next node has acknowledged receipt. The next node will be the final destination if the data is being sent intra-region, otherwise the DTN node will send the data onto the next region should a link into that region exist.

It is worth noting that, while long-distance wireless communication technologies may be used by certain types of mobile SMC (for example, GPRS) which would lessen the need for such mechanisms, these technologies do not guarantee network connectivity. That is, even in scenarios where such technologies were to be included in mobile components, there still exist periods of time where a delay-tolerant mechanism between regions is beneficial.

V. Implementation Details of Core Services

Implementation of the wide-area SMC involved tying together a number of largely unrelated codebases. For example, code which was capable of using the underlying SLP libraries is also required to use the Siena libraries to generate *newSMC*, *newMember*, *purgeMember* and *purgeSMC* events. Likewise, code is required to translate Siena events into a form which Ponder2 can understand.

Each of the primary components implements the ManagedObject interface to allow control via the Ponder2 policy service, and is initiated by the policy service using its XML markup to define commands. This offers the possibility of reconfiguring components via the policy service at runtime.

This section details the implementation-specific decisions for each of the core SMC components.

A. Event Bus

The implementation of the compact event bus we built for smaller environments (e.g., PDAs) was based on the freely available Siena codebase, but event types in the compact event bus do not exactly match those from Siena codebase.

It is ideal for the purposes of integration if events adhere to the same type descriptions in all mechanisms purporting to be an event bus, thus allowing an easier transfer of events between environments without translation. It would not take much effort to re-engineer the smaller event bus to use the Siena types, thus leading to easier communication between the two systems.

The wide-area SMC uses the Siena codebase without modification. A Managed Object component controlled by the policy service instantiates the event bus, and allows the policy service to subscribe to events. This Managed Object is also responsible for translating events to and from the XML format Ponder2 requires.

B. Discovery Service

The discovery service, as a mechanism contacted by services and devices rather than one which actively seeks new devices is naturally designed as a directory service for lookups, as discussed in Section II.

SLP provides a simple mechanism for locating such directory services. SLP entities use a known IP multicast address to query DAs without the UAs or SAs requiring prior knowledge of the DAs; this allows for DAs to be added to a local-area network, boosting scalability while not interfering with the operation of the other agents. The use of multicast, however, suggests that SLP works within multicast regions, but not across regions.

There are two potential solutions to this problem:

- UAs and SAs can be pre-configured to know the location of certain specific DAs, thus using unicast links to operate across multicast regions. In this situation, the SLP entities operate like a traditional directory service.
- Multicast regions can be explicitly connected by a network engineer, after which normal SLP functionality can come into play.

DAs store service advertisements, such as services local to an SMC like the event bus, and global services such as the discovery service itself. Thus, within an SMC there exists at least one directory within which all local services are registered. Other directories can be added transparently to allow scaling, should an SMC require it.

SLP scopes are used to restrict the visibility of SMC components. Each SMC is a member of two scopes, *SMCDiscoveryZone*, and a scope name derived from that SMC's ID and type (e.g., *Hospital:0013A90F201C*). Only discovery services advertise themselves within the discovery zone (and thus globally visible to all other discovery services), with all other services contained within the SMC-specific scope (only visible locally).

This is suitable for SMCs located within the same LAN, assuming multicast is available. For example, we have often discussed a hospital SMC consisting various wards, offices, beds, etc.

To cross multicast regions SLP nodes can be made aware of other directory services at configuration time (and, in some implementations, runtime). Unicast links can be used to talk to these other directory services, allowing access in a much more traditional directory-lookup manner.

The SLP Managed Object is configured with names or types of other SMCs which it should attempt to connect to; when one of those is spotted within the *SMCDiscoveryZone*, the Managed Object creates a new TCP connection to exchange information required to initiate policy-level interactions not otherwise stored in the advertisement, prior to the generation of a *newSMC* event within each SMC (see Section V-C).

While the personal-area discovery service worked on the principle of periodic broadcasts and unicasts to ensure that stale membership data did not linger, SLP advertisements are tagged with a lifetime which can be used to achieve the same effect, provided services are willing to re-advertise themselves periodically. Thus, services can fail (or be removed without notification) and related state will eventually be removed.

C. Policy Service

Ponder2 is used as the policy service as before, but we're now using Alberto's code **??** to form interactions between the SMCs. This opens up the possibility of loading missions, etc, onto child SMCs.

Appropriate Managed Objects for the event bus and the discovery service have been built to allow policy control of these components.

D. Delay-Tolerant Networking

While a DTN implementation is available [13], it would have been time consuming to engineer Java code to reliably interact with the DTN libraries. Instead, an entirely custom implementation in Java was built which obeyed the semantics we required of the asthma scenario, but which will not interoperate with other DTN implementations.

Data in transit is serialised and stored in non-volatile storage. Each bundle of data passing through a DTN network is uniquely identified by the destination region plus a unique identifier maintained at the current node. Thus, a node forwarding data from multiple nodes can deliver data in the order it was received, and no mechanism for globally unique identifiers is required.

Our DTN implementation obeys custody transfer semantics; that is, that as data is moved upstream the next intermediate node acknowledges custody of data in transfer, thus allowing the previous node (the original source or another intermediate) to immediately free up space.

Nodes are addressed as region:region_id; in the ZigBee realm, this would translate to *Patient_SMC_ID:Device_ID*, and in the wide-area may translate to *NHS:gp0001.nhs.sco.uk*.

E. Tying SMCs together

Based on the descriptions provided in the preceding sections, the following sequence of events takes place to pair two SMCs, as depicted in Figure 2:

1) The DiscoveryService Managed Object of an SMC periodically queries the DiscoveryScope SLP scope for other SMCs with which communication might be initiated; SMCs can be set up to either connect to other SMCs by type, or by a specific name. On location of a valid SMC, the initiating discovery service Managed Object will open a TCP connection over which any additional information required to initiate policy-level interaction will be exchanged (for example, Ponder2 requires an object ID (OID) in the remote policy service to initiate any kind of interaction).
2) Provided both SMCs agree to continue, each discovery service independently generates a "*newSMC*" event which carries all the appropriate information: remote SMC name, type, DTN region, policy OID.
3) The event bus delivers this event to subscribers; one such subscriber must be the policy service.
4) The policy service in the SMC which did not initiate contact will then initiate the high-level connection with the initiator using the information carried in the "*newSMC*" event.

This final stage, incurring actual policy-level interactions between SMCs, uses the work we presented in [14].

VI. Experimental Setup & Proof of Existence

The asthma scenario [2] was designed as a useful example within the e-health context to demonstrate SMC interactions and the scalability of the SMC management pattern. In summary, the asthma scenario calls for asthma inhaler usage to

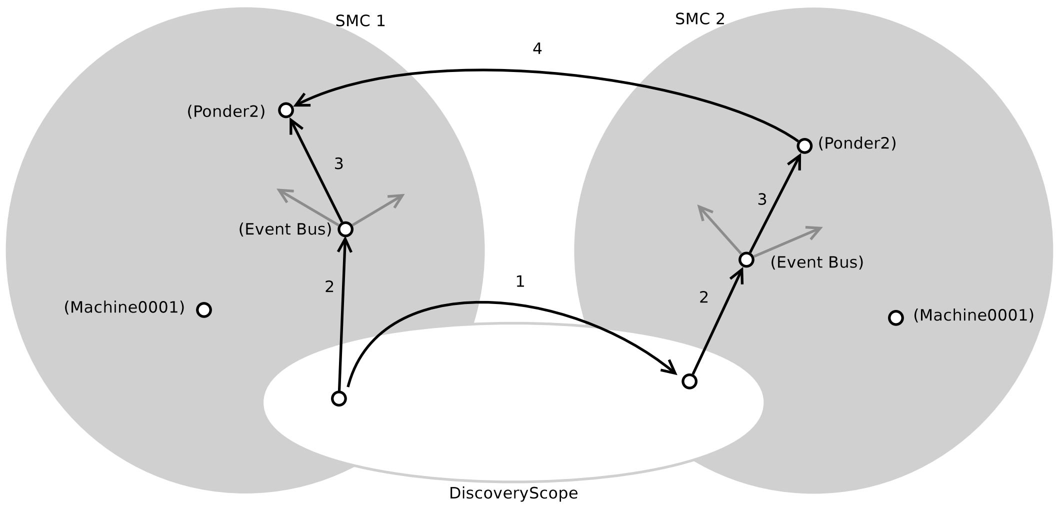

Fig. 2. Setting up policy-level interactions.

be logged at the patient, and periodically transferred to that patient's GP surgery. Data is transferred in a delay-tolerant manner, and is tagged with both timestamp and location[1], to be later accessed by a healthmap application.

Figure 3 shows the hierarchy designed to run across 49 virtual machines. Each virtual machine is a Xen guest domain, running kernel version 2.6.11 and configured with 180MB RAM. Each SMC is using the same wide-area SMC codebase, though a real-life implementation would utilise the personal-area SMCs previously developed and methods for interfacing the two realms (Section III). The SMCs configured and connected to each other as follows:

- 3 hospitals, connected to each other.
- 3 GP surgeries for each hospital (totalling 9)
- 2 home gateways per GP surgery (totalling 18)
- 1 patient per home gateway (totalling 18)
- 1 ambulance, configured to belong to one of the hospitals.

Thus, the 49 SMCs form a hierarchy, with multiple mobile SMCs interacting with stationary SMCs.

Patients disappear and reappear regularly, to simulate the roaming ability of the mobile SMC. Note that all patients may come into range of the single ambulance inhabiting this world, or their parent 'home' node. In real life, a patient might come into range of any other node listed here.

A. Description

The "NHS" region is the fixed network infrastructure in the simulation; all nodes within this region are assumed to be stationary points which do not move, and do not disappear. Indeed, in a real situation, the only nodes within the NHS likely to change regularly are the home gateways.

[1]Location can either be an accurate location, via GPS, or a location later appended by the GP surgery based on the postal code region in which the surgery operates. Datasets mapping postal codes onto geographic coordinates are freely downloadable (e.g., `http://www.npemap.org.uk/data/`, available as of 07/Aug/2007).

Data is transferred via the DTN from the PATIENT_MOBILE region into the NHS region, where it is routed to the appropriate GP surgery; the data to be shipped into the NHS region is tagged not only with the target region, but also with the target host. We may assume that the patient's SMC is loaded with the hostname and region to send data to when it is loaded with a particular mission.

The simulation runs by instantiating a new "patient" SMC at a certain time. Days run 24 minutes long, so in each 24 minute cycle, a patient SMC will be destroyed after 8 minutes, and restarted after 18 minutes (similar to a fairly normal commute-work-commute cycle). For the duration of the simulation, a separate process generates the false inhaler data; this ensures that data is present for the DTN layer to transfer when the SMC is restarted, in much the same way that data would be available when the SMC came into range of to another SMC.

During the 'hours' in which the patient is roaming the controlling script may randomly bring the patient back into existence for a short time, but this time configured to connect to the ambulance in the environment. The random chance is approximately 0.2%. This allows for data to be transferred to the NHS_MOBILE region by the DTN, prior to the patient "disappearing" again. The ambulance, while technically a mobile unit, does not appear and disappear from the simulation in this fashion, but stays in place to demonstrate the DTN routing appropriately between regions.

In reality, a patient SMC would be configured to talk to home, or the surgery, or the ambulance, etc, in some order of preference, rather than the reconfiguration which is required here. Generally, the patient SMC would only be in wireless range of one of these entities.

Data is stored at the GP surgery in a database; for simplicity, each GP surgery node runs a mysql database, which the application code accessed via JDBC. Data is stored in the form `(uid, patientId, timestamp, arbitraryData)`. We assume there must exist other tables

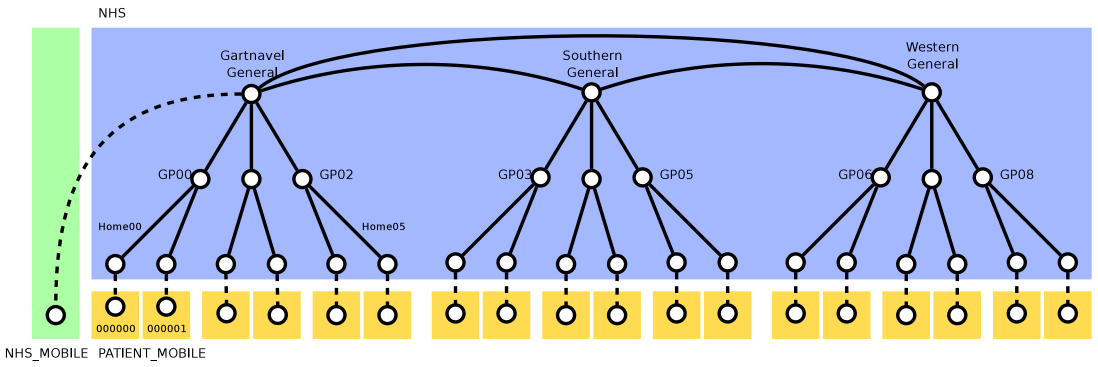

Fig. 3. Simulated SMCs mapped onto DTN regions.

holding other data about the patient, in particular one indexed by patientId and holding personal data, e.g., `(patientId, foreNames, surName, DOB, address, ...)`.

"arbitraryData" is ignored in the simulation, but it could be used to store the current geographical location logged at the same time as inhaler usage if we assume the patient is also carrying a GPS receiver as part of their SMC. Otherwise, given that GP surgeries operate within certain postal code regions, we can map usage statistics onto a specific location. This method, would reduce cost of such a device, but provides a more coarsely-grained representation of inhaler usage. (In particular, while it may show longer-term trends such as greater total inhaler usage per person near to a city, it may not be able to accurately display city centre inhaler usage during rush hour, for example.)

B. Running the SMC

Each SMC is represented by a virtual machine. To configure each VM including its SMC, we requires three independent configuration files:

1) One config file for the Xen VM itself, configuring its filesystem, IP address, and volume of RAM.
2) One script to start up the SMC. This script ensures that the environment is sane, restarts running daemons (such as slpd, and rmiregistry for the policy service), clears up stale data (e.g., DTN data, if starting afresh), and kick starts the policy service. These scripts are named "start_hospital.sh", "start_patienthome.sh", etc, though the only variation between these scripts is in the naming of SLP scopes. Patient SMCs run this script through another script which handles the regular appearance and disappearance of the SMC while still generating data for the SMC next time it is started. Most configuration details come from the configuration passed to the policy service.
3) The policy service reads in ./resource/boot.xml. This file starts up and configures core services, and defines key characteristics of the SMC, such as the SMC ID, type (profile, in Ponder2 parlance), DTN settings such as region, which regions are valid for upstream transmission, etc.

Once booted, each virtual machine represents an SMC, and is capable of connecting to other SMCs via the SLP mechanisms and interacting with those SMCs.

For the purposes of emulation, SMCs are similar enough that each of the VM configs, startup scripts, and boot.xml files were batch generated, with a symlink created from boot.xml to the configuration appropriate for the host.

VII. Visualisation of Data

The purpose of data generation and collection in the environment we have considered is to facilitate visualisation of stored data. This section covers briefly some of the software platforms available which may be capable of handling such visualisation tasks.

The healthmap we envision is not intended for the visualisation of real-time data, but rather for the analysis of data gathered over time. Given that SMCs must be instructed to collect data according to certain variables (e.g., characteristics to monitor, frequency of reading, etc) in advance of any actual visualisation, we expect the healthmap to be flexible within the dataset available to it. In particular, transferring data across a DTN may result in the situation where timestamped data arrives out of order, so the visualisation much be redrawn. We do not view this as an interactive application capable of "pushing" missions out to patient SMCs.

There are various visualisation tools (commonly, Geographic Information System, or GIS) available for use, each of which allows some level of configuration to allow the customised display of various datasets.

Quantum GIS is an open-source, 2-dimensional GIS [15], capable of generating custom overlays for the visualisation of datasets. However, the software does not come bundled with geographic data while other applications do.

Google Earth is a commonly used GIS available free for use on multiple platforms. Unfortunately, to use Google Earth as a GIS platform requires purchase of the "Pro" version, at a cost of $400 per annum.

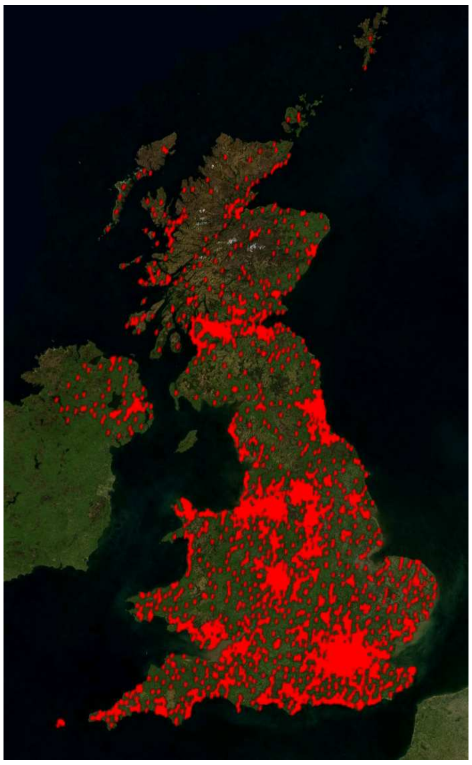

Fig. 4. UK map displaying postal code regions.

NASA World Wind (and the recently released World Wind Java SDK) is a strikingly similar project to Google Earth, in that it provides a 3-dimensional globe which can be zoomed, rotated, spun, etc. World Wind provides an open platform from which all manner of visualisation tools can be constructed. The new Java SDK should allow for an open, cross-platform approach.

Perhaps the easiest to integrate software currently available is the NASA World Wind Java SDK [16], released May 2007.The Java SDK would offer the easiest, most malleable method of building a custom visualisation system. Figure 4 demonstrates a basic overlay of postal code regions mapped onto their geographic coordinates using NASA World Wind; a visualisation using the Java SDK would likely be easier to build, but would appear similar to the image shown here.

VIII. Current Status

Each of the various core SMC components as described in this document have been built and tested. The delay-tolerant networking code has also been built and tested. Each of these components has been used within the context of the network emulation we constructed across multiple virtual machines.

The simulated network did not run to the full 49 VMs, but instead ran to 33, given the time constraints placed on generating the simulation. There is no reason why 49 SMCs would pose any problems, given the working status of the simulation at 33.

Having successfully run this emulated SMC infrastructure, the simulation of data generation for the purposes of display on a healthmap is close to complete. Additional work is required to retrieve data from databases and draw data onto a map surface to demonstrate a simple healthmap application.

Worth serious consideration is a more efficient method of storing of data across GP surgeries, in particular one designed with NHS concerns in mind.

Further, the real-world equivalent of this network would see mobile units using a more lightweight codebase to the wide-area SMC codebase they are using here; each node in the simulation uses the same codebase on the assumption that the core services are the same no matter the size of the SMC, and that the building of a gateway between the mobile realm and the stationary realm is an achievable task.

IX. Conclusion

We have built SMCs suitable for wide-area environments which behave differently to the body-area SMCs previously explored for monitoring patient state. These wide-area SMCs expect the presence of IP-based networks.

To bridge the two very different worlds, we have considered delay-tolerant networking for data transfer, and how the behaviour of the two discovery services can be mapped to avoid problems. The event bus is designed such that events crossing the boundaries can still be forwarded in either realm, and the same policy service is used in all SMCs.

Thus, we have constructed SMCs to operate in two very different environments which should be able to interoperate with each other, autonomously forging connections and managing themselves.

Our work leads nicely into the concept of a "healthmap", which would be capable of building visualisations of data stored at GP surgeries.

References

[1] E. Lupu, M. Sloman, N. Dulay, and J. Sventek, "AMUSE: Autonomic Management of Ubiquitous Systems for e-Health," http://www.dcs.gla.ac.uk/~joe/auxiliary/files/amuse-CfS-final.pdf, last accessed 07/Aug/2007.

[2] S. Strowes, "Health Map Scenario: Asthma," http://www.dcs.gla.ac.uk/~sds/papers/sds_healthmap07.pdf, University of Glasgow, Tech. Rep., Febuary 2007.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, Aug. 2001.

[4] S. Strowes, N. Badr, S. Heeps, E. Lupu, and M. Sloman, "An Event Service Supporting Autonomic Management of Ubiquitous Systems for e-Health," *ICDCS Workshops*, pp. 22 – 27, 2006.

[5] O. Foundation, "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map," http://tools.ietf.org/rfc/rfc4510.txt, June 2006.

[6] "Core Java: Java Naming and Directory Interface (JNDI)," http://java.sun.com/products/jndi/, accessed 07/Aug/2007.

[7] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," http://tools.ietf.org/rfc/rfc2608.txt, June 1999.

[8] "OpenSLP," http://www.openslp.org/, accessed 07/Aug/2007.

[9] Y. Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright, "Simple Service Discovery Protocol/1.0," Oct 1999, expired Apr 2000. http://tools.ietf.org/id/draft-cai-ssdp-v1-03.txt, accessed 08/11/2006.

[10] S. Helal, "Standards for service discovery and delivery," *PERVASIVE Computing*, vol. 1, no. 3, pp. 95 – 100, July – Sept 2002.

[11] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," in *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*. London, UK: Springer-Verlag, 2001, pp. 18–38.

[12] F. Warthman, “Delay-Tolerant Networks (DTNs); A Tutorial,” May 2003.

[13] “Delay Tolerant Networking Research Group,” http://dtnrg.org/, accessed 07/Aug/2007.

[14] A. E. Schaeffer-Filho, E. Lupu, N. Dulay, S. L. Keoh, K. Twidle, S. Heeps, S. Strowes, and J. Sventek, “Towards Supporting Interactions Between Self-Managed Cells,” *1st Internation Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, vol. 0, pp. 224 – 236, July 2007.

[15] “Quantum gis website,” http://qgis.org/, accessed 7th Aug, 2007.

[16] “Nasa world wind: Java sdk,” http://worldwind.arc.nasa.gov/java/, accessed 7th Aug, 2007.