Presented at Second International Conference on the Advanced concepts in: Applications of Digital Information and Web Technologies, ICADIWT '09. pp494 - 499, 4-6 Aug 2009

Discovering the Home : Advanced Concepts

James Dooley
Intelligent Inhabited Environments Group, University of Essex, UK.
jpdool@essex.ac.uk

Vic Callaghan
Intelligent Inhabited Environments Group, University of Essex, UK.
vic@essex.ac.uk

Hani Hagras
Intelligent Inhabited Environments Group, University of Essex, UK.
hani@essex.ac.uk

Phil Bull
Customer Networks Group, British Telecom, UK.
phil.bull@bt.com

Abstract

In this paper we present the experimentally validated discovery methods of our "Nexus" framework for use in home networks. These methods allow "client" software applications to dynamically search and access an entire set of distributed software resources in a home network in close to real time. The novelty of this framework is two-fold; firstly, no prior knowledge of the network is required by the client application. Secondly, the network itself does not need to actively maintain state by using servers/registries/caches.

1. Introduction

With an increasing deployment of heterogeneous electronic devices and associated soft resources (media, data files, processes, etc) into our homes, there is an opportunity to compose applications from these functional resources at runtime. This applies not only to a small sub-set of application domains (entertainment or lighting automation), but to nearly all functionality that exists in our homes (safety/security, communications, air conditioning, monitoring, lifestyle, etc). This has been the vision of pervasive/ubiquitous computing research [1][2] for some time now.

To illustrate this, consider the "jukebox" example where an application discovers hardware (speakers, GUI displays, person tracking) and software resources (codecs, audio files/playlists) necessary in the "Home Area Network" (HAN) to fulfill user requirements. Such an application would retrieve the audio files/streams (source) according to user preferences, pass them through the necessary codecs (processing) and attempt to stream the audio to speakers and present a GUI to the user (output/render). The speaker outputs and GUI could be dynamically changed in response to users moving through the home environment (using the person tracking component).

In order for this to be realised, unified methods/models are required to allow the entire set of resources available in the HAN to be discovered and utilised by software across differing application domains. Furthermore, these methods/models must operate under the constraints of real world home environments.

In [3] we presented the overall conceptual view of a unifying middleware framework we call "Nexus"[1], in which all software resources in a network can be uniquely identified, discovered, located and described. In that paper we addressed the basic requirements of discovery (discovering all the entities in a HAN) and lookup (finding the location of a specific entity given its unique identity). The experimental results we reported demonstrate an improvement in performance time by two orders of magnitude when compared to other middleware solutions. In this paper we provide more detail of the nexus framework, in particular we describe :

1. The "Entity Model",
2. The Network model,
3. An overall software framework for deployment,
4. Advanced distributed search methods.

In earlier work, we have shown that, current methods are unsuitable for time efficient discovery of distributed, high volume, resource deployments [3]. Also, current approaches do not offer generic abstractions which are required to allow a multitude of resource types to be used across differing application domains.

2. Functional Requirements

Formally, a HAN is composed of a set of software entities (**SE**) distributed over a set of physical devices (**D**). Where an individual (and uniquely identified) entity (**ε**) is hosted by an individual device (**d**). Additionally, the set of entities which a single device (**d**) hosts is labelled (**E**), and every (**E**) is a subset of (**SE**). Inversely, we say that (**SE**) is the superset of entities in the HAN.

From the superset of entities (**SE**) in a HAN, discovery is the process of retrieving a subset that

1 : From the word meaning "a connected group"!

Presented at Second International Conference on the Advanced concepts in: Applications of Digital Information and Web Technologies, ICADIWT '09. pp494 - 499, 4-6 Aug 2009

fulfills certain criteria. The exception to this rule is the **DISCOVER_ALL_OP** operation which unconditionally selects all entities (otherwise known as an enumeration). When there are large numbers of entities in a HAN, this is clearly going to take a (relatively) long time. At the other extreme is the lookup operation (**DISCOVER_OP**) which specifies the unique identity of the single entity to be retrieved (this however requires prior knowledge of the entity ID).

There is a very obvious need for additional operations which can find entities based on certain specifiable criteria (and fills the gap between specificity and enumeration). More formally (and forming our first requirement); we need a standard set of selection functions to map members of the set (**SE**) into a "results" set (**RE**). Where :

$$r\varepsilon \in RE,$$

then,

$$f : RE \rightarrow SE,\ f(r\varepsilon) = r\varepsilon$$

That is; the function f is the inclusion map of (**RE**) into (**SE**).

Naturally; if we are selecting entities based on specifiable criteria, each entity must have an information space against which we can evaluate our selection functions. This forms our second requirement; Each entity must have an associated information space against which selection functions can be evaluated.

These two requirements are addressed in sections 4 and 3.1. respectively.

3. Model

In this section we describe the Nexus model, with specific emphasis on the concepts needed to understand the discovery methods presented in the next section. With regards to the second of our functional requirements, the information space is described as a set of facets in [3.1.Entity Model].

3.1. Entity Model

In the Nexus framework, an individual software resource is known as an "Entity" [Figure 1] independent of what it exactly is (this is similar to the concept of an object in object-oriented computing). This abstraction forms the basis of our "Entity model" and allows equal treatment of files, processes, humans, agents, devices, etc.

Each individual entity has a unique immutable namespace qualified identity formatted as a URN (universal resource name). Additionally, each entity is self describing and presents its information space to the network. This information space is composed of individual XML encoded knowledge representations called "Facets". Each facet declares a "knowledge language" to which it adheres, this is enforced through XML schema (XML documents can be validated against XML schema). An individual facet is identified by the URN of the language that it obeys, therefore a specific item of information can be identified as :

"<**Entity-ID**>.<**Facet-Name**>.<**XPath**>"

Where an XPath is a standard (W3C) way of querying a single XML document for content.

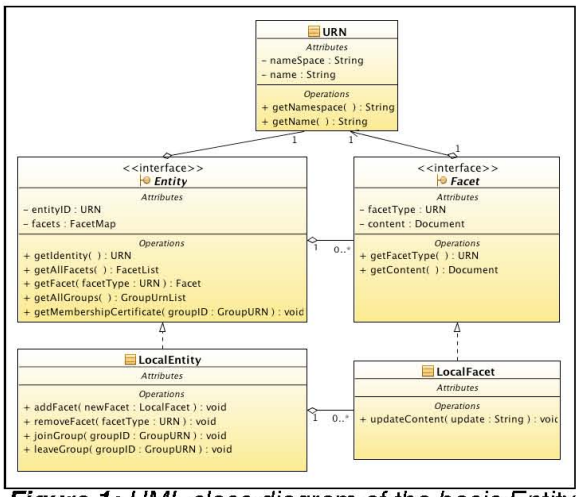

Figure 1: *UML class diagram of the basic Entity components.*

[Figure 1] shows the Entity and Facet interfaces in UML and the concrete "local" implementations. These concrete classes are the entities themselves, which specific entity implementations can extend (or simply populate with the necessary facets).

Not shown are the concrete "proxy" classes which network clients use to represent the remote entity and Facet objects (and which hide the network layer interactions).

3.2. Network Model

Unlike other middleware models, nexus clients directly find and interact with entities (no middle-man). This is in contrast to the typical method of dynamically locating "servers", then programmatically interacting with those servers to access items (for example finding a multimedia file in UPnP requires that the "media servers" are first located, then the "media server interface" be interacted with on each of those servers to find the actual files). The result is that fewer interactions are required by fewer network components, thus improving task performance time and reducing the number of possible failures that could occur.

On first inspection this would suggest that the network model employed is flat, making it hard to manage. To combat this, the Nexus model supports native grouping. A group is itself an entity and can therefore be interacted with in the same ways (has identity, has an information space, can be dynamically discovered, etc.). Each group has a policy that governs its specific behavior. The two most notable properties that are enforced through this policy are :

Presented at Second International Conference on the Advanced concepts in: Applications of Digital Information and Web Technologies, ICADIWT '09. pp494 - 499, 4-6 Aug 2009

1. **Embedding :** Governs whether or not sub-groups can be embedded within this group. The consequence of allowing this is the formation of a hierarchical tree with a root group.
2. **Duplication :** Governs whether or not an entity can exist in multiple subgroups of the same tree. If this is not allowed, the root group can be considered a superset, with each sub-group forming a sub-set.

The combination of these two behavioral modifications allow the single group concept to be treated as strict sets, single groupings or hierarchies.

When an entity successfully joins a group, it is given a cryptographically signed (currently using an 512-bit RSA algorithm to create an SHA-1 signature) membership certificate. This certificate declares that the entity is a valid member of the group for a certain time period (a quantum). Third parties can validate the certificate using the public key of the group which is freely available (published in the groups information space).

Multiple root groups may exist in a single HAN, to fulfill different requirements. A single entity can belong to as many different group trees as is necessary (but can only be duplicated in the same tree if permitted by the group policy).

As an example, consider a group which reflects the physical layout of a home. The root group may represent the home itself, with several sub-groups representing the rooms/spaces in the home (such as kitchen, bedroom, hall, etc.). To go further, each room/space could be split into logical sub-groups representing areas in a room. Each group can then have entities as its members. This arrangement provides for a very simple method of context awareness (to discover which display to use for a user, simply discover a display entity in the same group as the user).

The final note to make about the Nexus network model is that there is no centralisation. If a device goes offline, only those entities it hosts will disappear from the network. When the device goes back online, they will again be made available. This highlights the dynamic nature of the network and can be considered self-healing. Furthermore the level of distribution has the very simple consequence that a HAN can be considered a parallel system. This is important when we discuss methods of discovery later on in this paper [4.Methods of Discovery].

3.3. Software Architecture

As already described, each device in the HAN hosts a set of entities. This is achieved through the use of a relatively low (ideally one) instance of the Nexus framework running per device which "contain" the entity objects themselves.

The Nexus framework has been implemented using the OSGi component framework [Figure 2] and tested to work on both embedded (Java ME) and desktop (Java SE) deployments. The "Nexus Core" bundle provides the necessary functionality and services to allow dynamic entity publishing and discovery. It is intended that applications and "entity farms" will be implemented as bundles that use the functionality exposed by the core bundle.

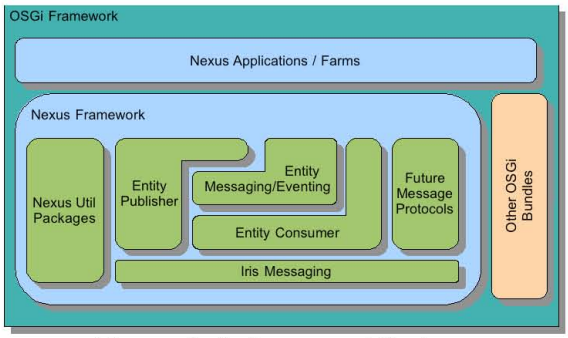

Figure 2: Software architecture.

While it is not hard to conceptually grasp what an application is, the concept of an "entity farm" needs some explaining; An entity farm is a factory that produces and configures entities. A farm directly interfaces with the entity publisher who's responsibility it is to publish all entities to the network. Many farms can exist within a single runtime (and be presented as entities themselves), each providing specific management of a logical group of entities. For example, a single runtime could be comprised of the following farms :

1. **Media farm :** That scans a directory on the local file system and wraps each media file (audio/video/image) as an entity (attaching suitable facets to describe each file).
2. **Agent Farm :** Hosting of software agents that interact with the network to fulfill some specific task. For example; a lighting agent could monitor light levels (read from light sensor entities) and adjust lighting entities as required in response to user preferences.
3. **X10 Farm :** Providing a wrapper for the X10 powerline control standard and allowing individual X10 devices to be presented to the network as entities.

By now it should be obvious that the mindset behind the model is to allow uniform network interaction with a variety of conceptual items (these three farms alone discuss agents, devices and media files). It is intended that any application can discover and interact with any item on the network without the need to use other protocols, it is however easy to get absorbed into an over enthusiastic habit of creating overly fine grained representation (as an extreme example; one could decide to represent each word in a document as an entity and present facets which specify where in a document they belong... this is obviously a miss use of the model).

Presented at Second International Conference on the Advanced concepts in: Applications of Digital Information and Web Technologies, ICADIWT '09. pp494 - 499, 4-6 Aug 2009

3.4. Message Exchange

The exchange of byte encoded messages over the network using TCP multicast and unicast allows for low latency, high throughput communications. Our earlier work has demonstrated that only a few milliseconds are required for a complete lookup operation to complete in a HAN containing 100,000 entities deployed over 7 devices [3].

The component responsible for message exchange in each Nexus runtime is called the "Iris messenger"[2]. This component [Figure 2] provides the basic functionality of message sending and delivery to other Nexus messaging components which implement specific protocols. Each iris message is typed (type specified using the OP_CODE field) and byte encoded, with structure as shown in [Figure 3].

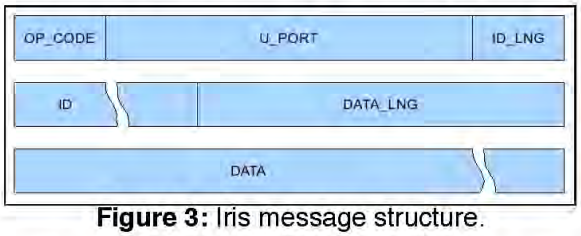

Figure 3: Iris message structure.

The packet sections of an iris message are as follows :

- **OP_CODE :** One byte that indicates the type of message (256 possible message types).
- **U_PORT :** 32-bit Integer (4 bytes) containing the unicast port of the sender to which TCP socket connections can be made (for iris messaging).
- **ID_LNG :** One byte containing the number of bytes in the ID section (maximum of 256).
- **ID :** This section contains the ID of the message conversation (UUID value[3]). Its length is defined by the value in ID_LNG section.
- **DATA_LNG :** 32-bit Integer (4 bytes) containing the number of bytes in the DATA section.
- **DATA :** This section contains message type specific data. Its length is defined by the value of DATA_LNG.

A typical UUID (forming the message ID) is 36 bytes long (32 digits separated by four hyphens), therefore a message with no data bytes (such as a DISCOVER_ALL_OP) would have a total length of 46 bytes. This is the typical minimum size of an iris message.

Much of the network related efficiency of Nexus in comparison to other discovery methods (UPnP, Jini, WS-Discovery, etc) is due to its use of byte encoded messages. This significantly reduces data transfer and marshalling/unmarshalling time.

2 : In Greek mythology, Iris was messenger of the gods and would travel with the speed of wind from one end of the world to the other.

3 : This will never exceed 256 bytes in length, hence the ID_LNG value only needs one byte.

4. Methods of Discovery

Our first functional requirement (identified in section 2), is the need for "discovery" operations which select entities in the HAN based on specifiable criteria. This section provides a description of the five Nexus framework discovery operations.

As discussed earlier in this paper, Nexus takes advantage of its distributed architecture and can be considered a parallel system. With specific regard to discovery, this means that the set of entities in an environment (SE) is not evaluated to generate the result set (RE) in a single runtime on a single device. Each discovery request is distributed to each Nexus runtime using multicast, where the discovery request is evaluated against the entities contained in that runtime and a partial result set generated. The results are returned to the requester (typically using unicast), where all the partial sets are combined to create the result set (RE). More formally; for a single discovery request and where (D) is the set of all devices in the HAN :

$$RE = \sum_{n=0}^{sizeof(D)} RE_n$$

The exception to this rule is a lookup operation (DISCOVER_OP) which will only ever result in a single result from a single container.

In the remainder of this chapter, we will discuss the individual discovery methods available in the Nexus framework. These methods form part of the Entity Resolution Protocol (ERP) along with entity announce and withdraw message exchanges (not covered in this paper). The purpose of this protocol is to resolve a network location URL (which may change over time) from an entity ID URN (constant over time). For example the entity identified by:

"**urn:nexus:myeg-1**"

could resolve to the URL :

"http://192.168.2.101:7071/entities/myeg-1"

From that URL, the entity state can be retrieved including the URLs of all the entity facets (XML).

4.1. Lookup (DISCOVER_OP)

A lookup operation (OP_CODE=0x04) takes a specific entity ID (URN) as a parameter and will solicit a response from the host container of that entity only. As the result set of this operation will only ever have a size of one, the result is sent as a multicast entity announce operation. This means that every Nexus framework in the HAN will receive the announce and add it to its local cache. The effect of this means that all the nexus frameworks become aware of the entity location for negligible overhead.

Presented at Second International Conference on the Advanced concepts in: Applications of Digital Information and Web Technologies, ICADIWT '09. pp494 - 499, 4-6 Aug 2009

4.2. Namespace Disc (DISCOVER_NS_OP)

A namespace based discovery operation (OP_CODE=0x05) takes a URN namespace as its parameter. Any entity IDs in that namespace will be considered a match to the discovery. A unicast connection is used to send all responses back to the requestor.

For example; [Table 1] shows a selection of example entity IDs and identifies those which would match if a namespace discovery operation was performed with the namespace parameter :

"**urn:nexus:resource:media**"

Table 1: *Namespace match results.*

Entity ID	Matches
urn:nexus:group:mg1	N
urn:nexus:resource:media:audio:song1	Y
urn:nexus:resource:media:video:song1	Y
urn:nexus:resource:file1	N

A standard set of namespaces exist to provide crude but fast static typing of entities. This is useful for applications which want to find a commonly understood "type" of entity such as software agents.

4.3. Group Disc (DISCOVER_GRP_OP)

As previously described in this paper, groups are a way of organising entities in the HAN. A group based discovery operation (OP_CODE=0x06) takes a group entity ID (in the namespace "**urn:nexus:group**") as its parameter and will yield a result set where all members belong to that group. As with the namespace based discovery operation, results are sent using unicast.

This form of discovery is appropriate for discovering members of a group when the semantics of membership to that group are understood. Going back to the example given earlier in this paper, if the semantics of group membership are :

"all siblings of a group in a hierarchy are physically co-located (eg in a kitchen)",

then group based discovery is useful to find physically co-located entities for purposes of context aware applications. More specifically, if speakers and a GUI are required in the same physical space by an application, group based discovery is appropriate.

4.4. Query Disc (QUERY_OP)

This operation (OP_CODE=0x07) is the slowest to evaluate, but the most flexible to use. As a parameter, this operation accepts a query script. The script language supports standard programmatic features such as constants, variables, arithmetic and functions. Each script is centered around the concept of testing each entity in the HAN against some conditions. If all those conditions pass, the entity is considered to match the query and becomes a member of the result set. A query has the ability to evaluate Xpath expressions against specific (named) facets in each entity. This provides the query mechanism with the ability of information space inspection for each entity. For those familiar with databases, it is functionally equivalent to an SQL query which can inspect table values. Thus we can treat the HAN itself as a large distributed database and select those entities which match certain criteria.

Going back to the "jukebox" example, consider each "music" entity (for example an mp3 file) to have a facet that captures the meta data of that song (similar to the content found in ID3 tags). Such a facet may appear as in [Figure 4][4] (each song would have different values in the respective positions).

```
<?xml version="1.0" encoding="UTF-8"?>
<fns:Facet
    facet-type="urn:nexus:facet:media:music"
    parent="urn:nexus:media:music:song1">
<ns1:MetaMusic>
  <ns1:SongInfo>
    <ns1:SongName>Radio GaGa</ns1:SongName>
    <ns1:Artist>Queen</ns1:Artist>
    <ns1:Released>1984</ns1:Released>
    <ns1:Genre>Rock</ns1:Genre>
    <ns1:Genre>Pop</ns1:Genre>
  </ns1:SongInfo>
</ns1:MetaMusic>
</fns:Facet>
```

Figure 4: *Example facet for a music entity.*

An application that needed to find all the music in the HAN by the artist "Queen", would generate a query script that evaluates an Xpath expression :

"ns:MetaMusic/ns:SongInfo/ns:Artist[text()='Queen']"

A complete script for this kind of query would look like [Figure 5].

```
#first some script constants
$baseNS = "http://nexus.org/schema"
$xp="ns:MetaMusic/ns:SongInfo/ns:Artist[text()='Queen']"
$facetType="urn:nexus:facet:media:music"

# define some namespaces for the xpath expressions
DefineNS ns=$baseNS/media

# perform some evaluation
EVAL XPATH($facetType, $xp)
```

Figure 5: *Example query script*

4.5. Enumeration (DISCOVER_ALL_OP)

This operation (OP_CODE=0x08) takes no arguments and yields a result set equal to the entity superset, ie :

RE = SE

4 : The xml namespace declarations are omitted for clarity.

Presented at Second International Conference on the Advanced concepts in: Applications of Digital Information and Web Technologies, ICADIWT '09. pp494 - 499, 4-6 Aug 2009

For this, each entity container will establish a unicast connection and send details about every entity it hosts. Quite obviously, when the number of entities grows in size the time taken to achieve this grows. While a useful tool for small deployments (<2500 entities) and development/debugging, its real world applications are limited due to the latency involved. This latency validates the need for the other discovery operations in terms of performance time, especially where the system is required to scale upwards to large deployments.

5. Discovery Method Comparison

To give an idea of how well the Nexus framework operates, we have deployed two devices in a 100MB network. The first is a 500Mhz ITX based "embedded" linux PC running a Nexus farm. The second is an Apple Mac Pro desktop computer running a Nexus client. We then carried out a set of 50 trials for each discovery method. The results in [Table 2] show the time required (as an average of the 50 trials in milliseconds, including network message exchange and all processing) to complete each method of discovery with varying numbers of deployed entities (in the range 10-1500).

Table 2: *Discovery "time to complete" (ms).*

	No. of deployed entities				
	10	**100**	**500**	**1000**	**1500**
Lookup	*2.36*	*2.4*	*3.28*	*2.98*	*2.64*
Namespace	*9.6*	*11.32*	*21.26*	*23.12*	*37.72*
Group	*13.38*	*12.8*	*22.92*	*24.18*	*20.84*
Query	*147.04*	*168.6*	*568.76*	*840.7*	*972.28*
Enumeration	*7.6*	*53.1*	*222.58*	*432.58*	*609.58*

Consistent with previously published data, a lookup operation completes in approximately constant time independent of the deployment size. Obviously, there is a small increase in completion time which is dependent on entity storage/indexing and the position of the entity in that store.

In contrast, the time required to complete enumeration and query operations is noticeably proportional to the number of deployed entities. For enumeration this is dependent only on the amount of time required to communicate the results (varied based on publisher device speed and network conditions) and can be considered linear. For a network with i devices and coefficient m for each device (the multiplier that accounts for a specific device speed and network latency involved in communicating results to the requester), the enumeration time will be approximately :

$$\text{enum\_time} = \text{MAX}((m_0 \times \text{size\_of}(E_0)) \ldots (m_i \times \text{size\_of}(E_i)))$$

We anticipate that there is also a "result integration" time penalty (proportional to the number of responding devices) which we hope to discover.

For the Query operation, the increase is dependent on the complexity of the query script and the number of facets that the script must be evaluated against per entity. We hope data from future trials will help formulate a query time complexity generalisation.

The two remaining operations (namespace and group based discovery) both display small variance (<12ms for group and <29ms) across the test range. These two methods appear to scale particularly well, but the completion time for group based discovery is however dependent on the number of groups to which each entity belongs.

6. Conclusions

In this paper we have extended our previous work and presented the basic concepts for the Nexus model through the entity model, the network model and the discovery mechanisms that are currently available. Furthermore we have given an account of the software architecture used to implement and deploy modular solutions using the Nexus framework.

We have validated this model with experimentation on a live network with both desktop and embedded systems and presented results which are consistent with our previously published data. Through this experimentation we have identified the need to optimize the way in which the framework stores and queries the entity facet content. Originally we used the xindice xml database, and migrated to the eXist XML database for reasons of query execution speed. However, we observed that when 1500 entities were deployed using eXist, the memory consumed by the Nexus framework was approaching 100MB! This is due to the use of the "in-memory" DOM model for XML representation. We now wish to implement a SAX model (freeing memory) and compare the impact of query execution.

We also wish to evaluate each of the query operations for comparison between themselves and comparison against other middleware solutions on a large scale deployment (40-100 devices).

7. References

[1] W.K. Edwards & R. Grinter. *"At Home with Ubiquitous Computing: Seven Challenges"*. Proceedings of the Conference on Ubiquitous Computing (Ubicomp 2001). Atlanta, GA. September 30-October 2, 2001.

[2] S. Helal, W. Mannm H. El-Zabadani, J. King, Y. Kaddoura & E. Jansen. *"The Gator Tech Smart House: A Programmable Pervasive Space"*. Computer. IEEE Computer society press. Vol.38, Issue 3, pp.50-60. 2005.

[3] J. Dooley, V. Callaghan, H. Hagras & P. Bull. *"Discovering the Home"*. To appear, International conference on intelligent environments (IE'09). Barcelona, Spain. July 20-21 2009.