Using Stigmergy to Co-ordinate Pervasive Computing Environments

Peter Barron and Vinny Cahill
Distributed Systems Group,
Department of Computer Science,
Trinity College,
Dublin 2, Ireland.
email: {Peter.Barron, Vinny.Cahill}@cs.tcd.ie

Abstract

Pervasive computing environments have proven difficult to develop in a form that supports the integration and organisation of devices and applications in a spontaneous and transparent manner. This is partly due to the highly dynamic and unpredictable nature of these types of environments, and is often further hampered by the limited resources found on devices. In this paper we present a highly decentralized method of organising the components of a pervasive computing environment, supporting spontaneous interaction between entities and providing robust-system wide behavior. Our inspiration for this work stems from nature and the observations made by the French biologist Grassé on how social insects co-ordinate their actions using indirect communication via the environment, a phenomenon that has become known as stigmergy. In the stigmergic approach there are fewer dependences between entities allowing for the incremental construction and improvement of solutions without adversely effecting the rest of the pervasive computing environment. The approach is encapsulated in Cocoa, a framework that supports the use of stigmergy to build self-organising environments that promotes the autonomy of entities. Experiences in using Cocoa *have shown that entities can be integrated into a pervasive environment in a spontaneous manner and that co-ordinated behavior can emerge.*

1. Introduction

Pervasive computing looks beyond the age of the personal computer to a time when every-day devices will be embedded with technology and connectivity. The goal for pervasive computing is to use these devices to transform physical spaces into interactive environments that can react in an intelligent manner, but do so in a way that is unobtrusive.

Developing pervasive computing applications to disappear into the fabric of our society requires the consideration of new and alternative approaches to system design. In this paper we introduce one such approach. We propose to meet the challenges of supporting the highly dynamic and unpredictable nature of pervasive environments by using swarm intelligence techniques, in particular, the powerful natural co-ordinating mechanism known as *stigmergy* [8]. This allows us to harness the robust, self-organising mechanisms observed in distributed natural systems such as social insect colonies.

The approach is encapsulated in Cocoa, a framework that exploits these techniques to support self-organising environments that encourage the autonomy of entities. Designed to both support and complement the use of stigmergy, the framework employs a distributed architecture organised in a peer-to-peer fashion. To ease the implementation and deployment we have also designed a programming abstraction encapsulated in a high-level scripting language. It generalises the methodologies used by social insects to construct a society of autonomous entities capable of responding to the environment in a stigmergic manner.

1.1. Stigmergy

In 1959, the French biologist, Grassé observed that social insects could co-ordinate their actions through the environment without having to directly communicate with each other. They do this using a phenomenon known as *stigmergy* [8]. He also noticed that the local interactions between insects resulted in the emergence of a strong colony-wide behavior. Holland et al also noted [9] that stigmergy provides a mechanism that allows the environment to structure itself through the activities of the entities within the environment. The state of the environment, and the current distribution of entities within it, determines how the environment and the entities will change in the future. This approach provides a robust, self-organising environment, which can co-ordinate its behavior in a highly decentralized manner. It is important to stress that individual entities have no particular problem solving knowledge, and that co-ordinated behavior emerges due to the actions of the society. It also worth noting that while no direct communication is used between individual entities, communication is still maintained through the medium of the environment.

The phenomenon of stigmergy has been used in a number of computer related projects, from robotics [9], to pattern detection and classification [3], to communication networks [11]. The goal of this work is to use the principles of stigmergy to create highly adaptive environments that allow for the incremental construction and improvement of solutions without adversely effecting the rest of the environment.

1.2. Road map

The rest of this paper is organised as follows: the next section investigates the use of stigmergy within a pervasive environment and outlines the main concepts used in the design of Cocoa. Section 3 and 4 describe the architecture. Section 5 details the implementation of the Cocoa framework. Section 6 describes the experiences of using Cocoa in a mobile, wireless environment. The paper closes by looking at some of the related work in this field of research.

2. A Proposal

The idea of simple behaving insects, with little memory or ability to exhibit any real intelligence, maps well onto pervasive computing where small devices with limited resources are spread across the environment. The indirect communication mechanisms harnessed by social insects provides a means for decoupling devices and applications. Having fewer dependences between components allows the overall system to be less fragile and more stable to disturbances in the environment. The system can grow organically and decay gracefully with the environment, as new devices are added and old ones upgraded or removed, without having an adverse effect on the overall system. The spontaneous interaction of devices and applications can be achieved as communication is done through the common medium of the environment. The use of stigmergy allows us to harness the robust, self-organising, coordinating mechanisms of social insects, which is perhaps the most desirable attribute for any pervasive computing environment.

We propose to use the principles of stigmergy to create a framework for pervasive computing environments, where context information of surrounding entities provide a common medium for the indirect communication mechanisms used by entities. The social insects observed by Grassé are represented as entities within the framework. An entity is a person, place, or object as defined by Dey [6]. Co-ordinated behavior in the Cocoa framework arises from entities observing their environment and reacting to the received context information according to some rules. Context information is any information that can be used to characterize the situation of an entity [6].

To apply the concept of stigmergy in a pervasive environment we must first define a process that adapts the behavior of individual entities to reflect changes in the local environment. Figure 1(a) represents the context of every entity in the pervasive environment at a particular time. It is the global context $C_G(t)$ of the environment. All information contained in $C_G(t)$ is not required by each individual entity, as the behavior of an entity is only dictated by the context of its local environment. Figure 1(b) illustrates a subset of the context information required by an entity. It represents the local environment and defines the entity's *contextual view* $C_{V_{e_n}}(t)$, as defined in equation 1. It holds all context information in $C_G(t)$ that is relevant to the situation of entity e_n at time t. An entity's context $C_{e_i}(t)$ is included in entity e_n's contextual view if the entity is within a certain proximity. The notion of proximity is used to define what the local environment of an entity is. This is captured in equation 1 where the function $L(e_i, e_n)$ is used to determine proximity and returns $true$ if entity e_i is within the required proximity of entity e_n.

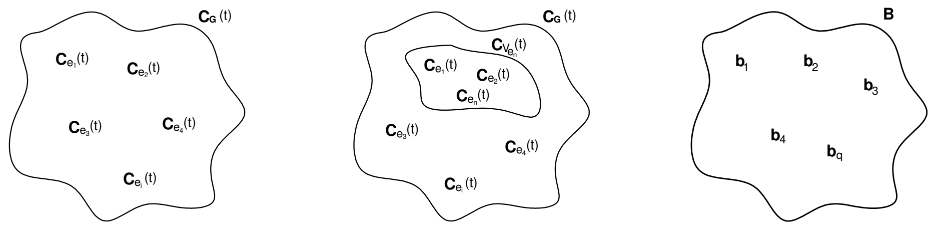
$$C_{V_{e_n}}(t) = \{C_{e_i}(t) : C_{e_i}(t) \epsilon C_G(t) \wedge L(e_i, e_n) = true\} \tag{1}$$

The behavioral set B, shown in figure 1(c), represents a finite set of behaviors that the entity can perform. For example, a light can either turn itself on or off, or a jukebox play music, pause, or stop playing. The behavioral set defines how an entity behaves in the pervasive environment. The last stage of the process dictates how individual entities behave. Equation 2 defines the function S for mapping $C_{V_{e_n}}(t)$ onto $\mathcal{P}(B)$[1]. $C_{V_e}(t)$ represents the collection of all contextual views. This function maps the entity's context information from the local environment onto a behavior, thus initiating a stigmergic response to the environment.

$$C : C_{V_e} \rightarrow \mathcal{P}(B) \tag{2}$$

The proximity function L, the behavioral set B, and the S function provide the three primitives used by the framework to define how individual entities behave in response to changes in the local context state of the environment. Over time system-level behaviors may emerge as different entities change their behavior in response to the changing behaviors of other entities.

[1]The power set of behavioral set B.



(a) Global context (b) Contextual view (c) Behavioral set

Figure 1. Using Stigmergy in a Pervasive Environment

3. Cocoa[2] Architecture

It has been necessary to develop technologies to both support and complement the use of stigmergy. The framework has been designed as a distributed architecture organized in a peer-to-peer fashion. Each node in the architecture represents an entity in the pervasive environment. The acquisition of context information uses a collaborative approach, whereby each entity acquires their own context and shares it with other entities in its locality. This distributes the process of capturing context information across the environment and at the same stage enhances each entities understanding of their environment. The primitives defined in the previous section are used to manage the collated context and define the mechanisms for determining entity behavior. The communication drivers provide a decoupled communication model that distributes events between entities. Binding the framework together is a scripting language. In the following sections we describe the the main components - context acquisition, stigmergy runtime, script - of each node, as shown in figure 2.

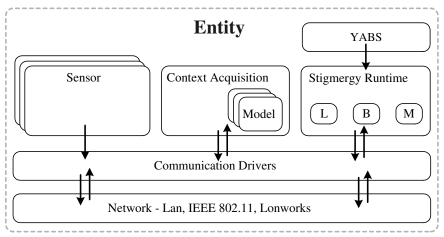


Figure 2. Cocoa Architecture

3.1. Context Acquisition

Sensors are an integral part of any context acquisition system and true to Mark Weiser's vision [21] we foresee an environment over which sensors are widely distributed. In the Cocoa environment each entity has a number of sensors associated with it. These are used to determine the context of the entity. An open interface is provided so that different techniques or *models* can be plugged into the framework to interpret the sensor data. It is possible to use simple IF-THEN rules, or sensor fusion techniques such as Bayesian networks, or any other technique that may be suitable.

This approach allows Cocoa to tailor the process of capturing context information for each entity. Over time an entity may change their model to suit different environmental parameters. For example a model may change if a person moves from his car to the office, or if the framework finds that there less computational resources available. The modular aspect of this approach ensures that entities are capable of adapting the process of acquiring context information to suit the environment.

What is important to note at this stage is that an entity only acquires context information about themselves and not other entities. The model determines what they are doing, where they are located, and any other information that may be useful in describing the entity's situation. The wider contextual picture is gained from entities sharing their context information with other entities. This they achieve by broadcasting their context to other entities in their local environment. The dissemination of the context information ensures that entities can build up a picture of their local environment, and hence gain a better understanding of it. It also distributes the process of capturing context information and provides entities with a shortcut in sensing their local environment.

3.2. Stigmergy Runtime

The component presented in this section provides the runtime environment for the entity. It uses the intermediate objects produced by the scripting component (YABS) to initialise the component. It is responsible for managing the entity's contextual view and for triggering the stigmergic re-

[2]CO-ordinated COntext Awareness

sponses of the entity. It provides the implementation of the three primitives - L, B and S - as described in section 2 and the runtime environment for this process.

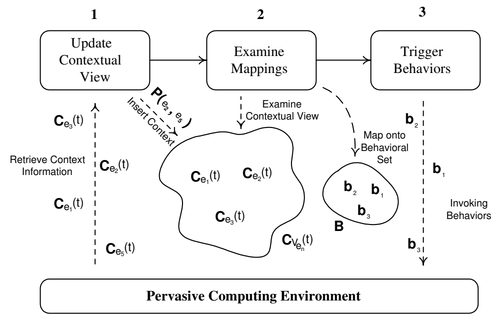


Figure 3. Stages used in runtime environment.

Each cycle of the runtime environment is composed of three stages as illustrated in figure 3. The first stage retrieves the context information from the environment and updates the entity's contextual view, $C_{V_{e_n}}(t)$. An entity's context information, $C_{e_n}(t)$, is included in the contextual view when L, the proximity function returns true. This indicates that the entity in question is within the proximity specified by the entity.

The second stage implements the S function, which consists of a series of mappings between the entity's contextual view and the behavioral set. The stage operates over a number of cycles gathering information from the entity's contextual view. At each cycle it propagates the state of the mappings and determines if one has been triggered.

The final stage is responsible for invoking the behaviors associated with any of the triggered mappings. It takes the implementation of the behavior and it passes the parameters indicated to it by the script. This may include values from the script or context information that needs to be derived from the entity's current contextual view. Once the parameters for the behavior have been determined then behavior can be invoked by the runtime environment.

3.3. Script - YABS [3]

Scripts are used to define the stigmergic responses for an entity and are responsible for generating the intermediate objects for the stigmergy runtime. The foundation for the language is built upon the three primitives: L, B and S. The language defines the proximity function. It specifies the behaviors that an entity is capable of performing and provides an method of mapping an entity's contextual view onto its behavioral set. We start by outlining the basic structures

[3]Yet Another Behavioral Script

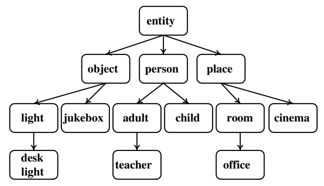


Figure 4. Script inheritance hierarchy.

and capabilities of the language and then continue to detail some of the more fundamental aspects of the language.

Overview YABS uses an interpreter that takes a text file containing a description of the desired behaviors. These are translated it into an intermediate form that can be executed by the stigmergy runtime. The behaviors described in the text file characterises how a particular type of entity behaves in the environment and may be reused by all entities of that type.

```
desklight extends object{...}
```

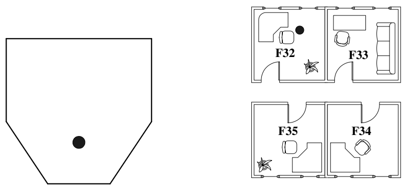
Listing 1. Declaring the start of a script.

The example shown in listing 1 defines a script for an entity of type *desklight*. Any desklight entity can use the behaviors described in the script to regulate how it behaves. It is possible to inherit behaviors from another script by extending a preexisting script. In the example above, the *desklight* inherits behaviors from *object*.

As you might expect inheritance relationships form a tree-like hierarchical structure. The inheritance hierarchy for this language, as see in figure 4, is influenced by the presence of four predefined scripts - entity, object, person, place - which enforces some structure on the hierarchy. The choice of scripts is influenced by the definition used by Dey [6] in defining an entity, where he defines it to be a person, place, or object. The inheritance relationships in the hierarchy are used within the script to determine the type of the entity in question.

Contained in the script are descriptions of the three primitives - L, B and S - introduced earlier in this paper. Together these primitives define how a particular type of entity behaves in the environment.

Proximity Function L, the proximity function can be defined as either a radius, polygon, or symbolic area around an entity. The context of entities entering this region will be inserted into the entity's contextual view. In the first example shown in listing 2 the proximity is set to a 5 meter radius around the entity. The next example uses pairs of co-



(a) Polygon (b) Symbolic

Figure 5. Proximity

ordinates to define a polygon: the unit of measurement is meters, and the reference point for the polygon is the position of the entity. The polygon defined by the sample code is illustrated in figure 5(a).

```
proximity(5) //cirle
proximity(-5,-5,-10,5,-10,20,10,20,10,5,5,-5) //polygon
proximity(F32) //symbolic location
```

Listing 2. Code for proximity functions.

Figure 5(b) shows the use of symbolic proximity, where a predefined area can be used to specify the proximity around an entity. This type of proximity is useful when there is a strong definable boundary, such as room, or building. It helps filter out interference from entities which are near by, but are not involved in the current situation i.e. are outside the boundary. The last example shown in listing 2 illustrates how this kind of proximity can be defined in the script, where it is set to an office called *F32*.

```
behavior on = "ie.tcd.cs.lighton"
behavior off = "ie.tcd.cs.lightoff"
```

Listing 3. Declaring behaviors.

Behavioral Set B, the *behavioral set* defines the set of possible behaviors that can be performed by the entity. The implementation of a behavior is not done in the script, but in Java following a particular API defined by Cocoa. The script declares behaviors that a particular type of entity can perform. In this instance the example shown in listing 3 indicates that the behaviors are either *on* or *off*, which in this case represents behaviors for turning a light on or off. When the behavior is invoked it executes the Java object that defines the specific behavior.

S Function The primary function of the script is to map an entity's contextual view onto the behavioral set. This is achieved by first defining context information that is of interest to the entity. These can be thought of as contextual predicates that are true if found to be in the entity's current contextual view. The code shown in listing 4 is one such example.

```
context vinnyperson
vinnyperson.person="Vinny"
vinnyperson.location="O'Reilly House, F32"
vinnyperson.activity=any
vinnyperson.time="lunch time"
vinnyperson.job="teacher"
vinnyperson.music="rock"
```

Listing 4. Declaring context information.

In this sample code the context called *vinnyperson* is declared. The keyword *person* defines the context *vinnyperson* as person with name of *Vinny.* It is also possible to identify a *place* or an *object* and by using the *any* operator you can specify any person, any object, or any place. The *location* keyword indicates a position or area that is of interest to the entity. It is possible to use GPS coordinates, relative coordinates, or symbolic information such as the *"O'Reilly House, F32",* as used in this example. The *activity* keyword defines what an entity is doing. This could be a person walking to work, a desklight turned on, or a printer printing. In this example the entity is interested in Vinny doing any activity. The *time* keyword indicates a period, or point in time. This can be specified as an absolute time such as *"Thu Mar 18 21:58:36 GMT 2004"*, or symbolic time such as *"lunch time"*. It must be noted that while symbolic context information can be used it needs to be agreed upon beforehand.

The script uses Dey's [6] concept of primary and secondary context information. Primary context information being the identity, location, activity and time of the entity, while secondary context information describes any other information which helps define an entity's situation. In the script secondary context information is declared by specifying any key/value pairing. In the coding sample above the *vinnyperson* context declares two such pieces of context information. The first describing the what job Vinny does and the second specifies what music he likes to listen to.

After declaring context information of interest to the entity it is time to use the information to map the entity's contextual view onto the behavioral set. In the example shown in listing 5 the mapping is accomplished when predefined context information is matched onto the entity's current contextual view. In other words when the context *vinnyperson* and *darkroom* are a subset of the current contextual view. On obtaining a match the behavior can then be triggered. In this case the *on* behavior is invoked when a person named *Vinny* is in a dark room.

```
map[ vinnyperson , darkroom ] onto {
   on()
}
```

Listing 5. Example of mapping statement.

4. Mapping

The previous sections outlined the main components of the Cocoa and demonstrated the process of triggering a behavior for an entity on encountering a specific situation described by fragments of context information. The recognition of this particular instance in time is often not sufficient to capture the broader sense of what has occurred and it is necessary to find a more expressive means of performing the mapping that can take into account what has occurred beforehand. Influence by the work of Allen [2] and that of Pinhanez et al's interval scripts [17], we look at a method which models the relationships between intervals of time to capture these type of situations.

4.1. Allen's Temporal Intervals

An interval of time is a length of time marked off by two distinct points in time, representing the start and end of the interval. In [2, 1], Allen introduced a model that made it possible to describe the relationship between two intervals of time. He showed that there are 13 such possible relationships, as summarised in figure 6.

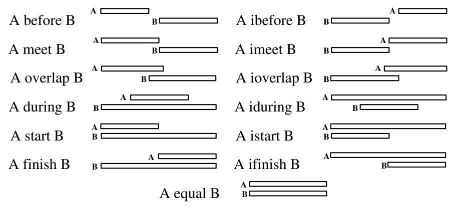


Figure 6. Interval relationships

Given any two intervals of time it is possible to use one of the relationships illustrated in figure 6 to describe how they are related. For instance, if we take a story such as the following:

> John was not in the room when I touched the switch to turn on the light.

we could use Allen interval algebra to describe the above story as follows:

```
S overlap or meet L
S is before, meet, is imeet,
or ibefore R
```

where S is the time touching the switch, L is the time the light was on, and R is the time that John was in the room.

The importance of Allen's work stems from its ability to provide a mechanism for describing the relationships between intervals without having to explicitly mention the interval duration or specifying the relationships between the intervals extremities. These characteristics are of significant value when it comes to describing the temporal relationships of events in a pervasive computing environment and especially when you consider the imprecise nature of these type of environments.

4.2. Scripting Temporal Intervals

Based on Allen's interval algebra the script uses the primitive relationships defined by Allen to describe temporal relationships between events in a pervasive environment. Entity behavior is then trigger on observing the events in the correct temporal sequence. The predefined contexts defined in section 3.3 are used as a means for describing events and hence the intervals of time for when these events are valid. The script specifies the relationships between intervals by defining a sequence of predefined contexts. The interval is active if the context is a subset of the entity's current contextual view. Once the intervals have occurred as indicated by the script the mapping can occur and behavior can be triggered.

```
map[ contextB , contextA ][ contextB ] onto { ... }
```

Listing 6. Mapping using temporal intervals.

For the purpose of further illustration we use an example to explain in more detail the use of Allen's interval algebra in the script. The sample code shown in listing 6 demonstrates the use of intervals in the mapping statement. It uses the predefined contexts *contextA* and *contextB* to describe two different intervals of time. The relationship between the intervals can be defined as *contextA start contextB,* as per Allen's interval algebra. The square brackets demarcate the start and end of the intervals, and defines the relationship between them.

In determining whether a mapping has been trigger the runtime environment investigates each subsequent contextual view to determine if intervals are active. An interval is deemed active when equation 3 is satisfied. In other words when C_i, a predefined context, is a subset of the entity's current contextual view - $C_{V_{e_n}}(t)$. Meaning that when the information specified in C_i is also contained in the context of an entity that is held in the entity's current contextual view. When the intervals are found to be active in the correct temporal sequence as that described in the mapping, then it is at this stage that the behavior can be triggered. In the example above, the interval *contextB* and *contextA* must be initially be active is the same contextual view and for subsequent contextual views until interval *contextA* becomes inactive, which at this stage will trigger the behavior.

$$C_i \subseteq C_{V_{e_n}}(t) \tag{3}$$

It is also feasible to use the other 12 relationships defined by Allen in the mapping statement. For instance, in the first example shown in listing 7 *contextA* is *before contextB*. The symbol *[ ]* indicates that no interval is active at this period of time. The other example shows the that *contextA overlap*s *contextB*.

```
//contextA before contextB
map[contextA][][contextB] onto {...}
//contextA overlaps contextB
map[contextA][contextA, contextB][contextB] onto {...}
```

Listing 7. Other mappings using intervals.

By using Allen's interval algebra we provide a more expressive means of performing mappings that take into account what has occurred beforehand and not just what has happened at a particular point in time. While it does increase the complexity of the script, it is felt that the increase expressiveness gained by the using Allen's interval algebra outweighs the additional difficulty in scripting the behaviors of entities.

4.3. Runtime

The stigmergy runtime monitors the state of the mapping in the second stage of the runtime environment. This stage operates over a number cycles, gathering information on when each interval starts and finishes. Over time the relationships between the intervals can be built up and it is at this stage that it is possible to determine whether the observations satisfy the constraints declared in the mappings. In the runtime environment this process starts by first taking the interval relationships defined in the mapping and dividing them into slots as illustrated in the example shown in figure 7. The process starts at the first slot, where the entity's current contextual view is used to determine if the next slot is valid. A slot is valid if the intervals defined in that slot are active and the remaining intervals are found not to be active. If this is the case then it is possible to move to that slot. To remain in this slot it must be valid in all subsequent contextual views, unless the next slot is valid in which case we move to that slot. If the slot is not valid then the process starts again at the first slot. On reaching the last slot the relationships between the intervals has been satisfied and the mapping can be triggered by the runtime environment, at which stage the whole process starts again.

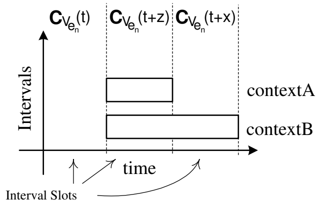


Figure 7. Interval slots.

5. Implementation

The current prototype of the Cocoa framework has been implemented in Java. The framework uses a modular design to aid both the extensibility and flexibility of the framework. This allows different components to be loaded at runtime depending on the entity and the environmental configuration. Each entity runs in its own computational space, although entities may be located on the same device they do not necessarily have to. The stigmergy runtime and the scripting language are implemented as described in previous sections.

For the context acquisition component a number of models have been implemented using the Cocoa framework. Models, as was described in section 3.1, are pluggable components that can be inserted into the framework at runtime for interpreting sensor data. The current generation of models typically aggregate the data from sensors to determine the context of entities. Other sensor fusion techniques can be used though at present have not been implemented.

In the current implementation it is also possible to plug-in different communication drivers to suit both the middleware requirements and network configuration. This makes it easier to modify the system to suit different environments. Communication drivers can be based on publish, subscribe mechanisms, tuple spaces, or other communication paradigms that provide a decoupled communication model. The Cocoa framework currently uses a driver based on Steam [13].

Steam is an event-based middleware service that has been designed with pervasive computing in mind. More specifically, it is intended for use within mobile environments using wireless ad-hoc networks. Steam exploits a number of novel techniques which allow it to operate successfully within these type of environments. In particular, it uses geographical information to limit the propagation of events through the environment ensuring scalability and the timely delivery of events. There is also no centralised components within steam and subscription to events are made dynamically to producers as entities move through the environment. Events can also be filtered on the proximity of one entity to another. The steam event service was chosen as it best suited the environment that Cocoa is presently being deployed in. A more detail description of steam can be found at [13].

6. Evaluation

Westland Row is a street located in the heart of Dublin, Ireland. The street is about 250 meters long, and accommodates a number of cafes, newsagents, shops, pubs, and a train station. It is a busy street, with commuters, shoppers, cars, and buses transcending it on a daily basis. A wireless ad-hoc network has been deployed on Westland Row, with a number of access points placed along the street. The access points form a sparse population of wireless network nodes and can be configured to create a variety of network models. The current model uses AODV [14] as the ad-hoc routing protocol for the network and uses a gateway node to access the Internet. The network is part of another project investigating the use ad-hoc networks in urban areas.

Westland Row provides both a challenging, and an interesting testing ground for evaluating pervasive computing applications. We use it as means of examining Cocoa and have developed and deployed a number of entities along Westland Row. To observe how the entities behaved we ran a number of scenarios. The following is a typical example: Peter and Vinny arrange to meet in Westcoast for coffee one morning. Coming from opposite ends of the street they meet at the entrance of the cafe, where they entered and have coffee. The following sections describes some of the key entities used in the scenarios and outlines our experiences in using such a framework.

6.1. Entities

Creating a pervasive environment with Cocoa starts by identifying the key entities in the environment. Then using YABS to define their behavior in response to the contextual stimuli of the local environment. The development process for an entity comprises three main steps: the first determines whether there is a suitable model for acquiring context information for the entity. If there is not, one needs to be created for the entity. The second step, requires the implementation of the behaviors for the entity, if such implementations are not available. The last step is to create a script that defines the behavior of the entity. The current society of entities are as follows:

Punter was one of the first type of entity to be developed for Westland Row. The entity represents a person on the street, whether they are shopping, having coffee, or commuting to work along the street. No behaviors were implemented for punter as Cocoa cannot change the behavior of a person. Even though Cocoa triggers no behaviors for this entity it is still necessary to represent the average person in the Cocoa environment, the reason being to allow other entities absorb their context. The context information typically comprised of the person's name, location, time, musical preferences. In the Westland Row environment the punter entity runs a on mobile device - PDA, laptop - associated with a person.

Siopa is the Irish word for *shop* and in the Westland Row pervasive environment is the type of entity used to represent the different the shops and cafes along the street. The current implementation of the siopa entity is quite basic, only capturing a very limited amount of context information about itself and has no behaviors associated with it. These type of entities run on PC104 devices embedded into Westland Row.

Firefox is a web browser that provides information about the environment. The current implementation uses the Firefox browser from the Mozilla Foundation to display information associated with entities. One behavior has been implemented for the firefox entity, called *display,* it opens a web page on the Firefox browser. The behavior is triggered when someone is nearby and when information is available to display. With the setup of the ah-hoc network on Westland Row it is possible to open pages located on the network, as well as those external to it. In the current setup firefox entities ran on the same device as punter entities.

Jukebox, as the name might suggest, is an mp3 player. Two behaviors have been implemented for the jukebox, called *play* and *stop.* In the current implementation the stop behavior is triggered when no one is the vicinity of the jukebox to listen to the music. The play behavior is triggered when a person is near the jukebox. The genre of music played depends on what the majority of people prefer to listen to. This is determined by observing the context information from punter entities, in particular the musical preferences of the entities. While this is the current behavior of the jukebox entity it can be modified to react differently to the environment. For instance, it could play different music depending on location, or it could always play what the minority of the people like to listen to. The jukebox entity ran on a laptop with speakers using the xmms mp3 player.

In all we deployed two punters entities, five siopa's representing some of the shops, cafes on the street, two firefox entities associated with each person, and a jukebox located in one of the cafes halfway down the street.

6.2. Experiences

From running a number scenarios on Westland Row we have observed some encouraging behaviors. The siopa entities along Westland Row remained passive to changes in their environment, which was as expected due to the current implementation of the siopa entity. Though the firefox entities, which were sometimes carried around by people would display information about the shops as they walk by. They would also display information to users about the songs being played on the jukebox entity. For instance in the scenario described in the previous section, the firefox entities displayed information on the music being played as Peter and Vinny entered the cafe. The jukebox entity, with its collection of music, would tailor the selection played depending on the users in it's vicinity. In the scenario above as Peter and Vinny entered the cafe shop the jukebox started to play more folk music to reflect the preferences of the users in the cafe.

It appears that entities can co-ordinate their behavior through the environment. However, in the current generation of entities co-ordination is restricted due to the limit number of entities involved, but we believe with a richer society of entities it would be possible to achieve much more. The benefit of using stigmergy was also observed through the indirect communication mechanisms used by the phenomenon. It was possible to add new entities, remove or upgrade old ones from Westland Row without adversely effecting of rest of the street. This allowed for the environment to be built incrementally and solutions to be improved on over time. There were fewer dependence between entities, which appeared to make the overall system less fragile and more stable to disturbances in the environment.

In developing the entities described in the previous section we noted that the scripting language successfully managed to separate the computational side of acquiring and managing context information with the compositional side of developing pervasive computing applications. The clear separation allows the developer to concentrate on implementing the behavior of individual entities rather than the lower system levels. We also observed that the traditional concept of a pervasive computing application shifts somewhat when using Cocoa, as the focus for development is centered on the entity and not solely on any particular application. The applications per say emerge from the pervasive environment as the entities move and reorganise themselves.

7. Related work

In recent times there has been considerable amount of work done towards supporting the development and creation of applications for pervasive computing environments. For example the TEA (Technology for Enable Awareness) [20] project uses a scripting mechanism to preform basic actions. The actions can be performed when entering a context, when leaving a context, and while in a certain context. Their framework concentrates on adapting the behavior of small devices such as mobile phones. The Aware Home Research Initiative [12] uses the Context Toolkit [7] to capture context information. The MUSE [4] infrastructure uses Jini based services in combination with bayesian networks to fuse raw sensory information into context information. There is also Pinhanez's interval scripting language [18]. Based on PNF-networks [15] it has help Pinhanez et al to create interactive environments such as SingSong [17] and It/I [16]. Pinhanez's PNF-networks are based on Allen [2] temporal intervals. Another project called RCSM (Reconfigurable Context-Sensitive Middleware) [22] provides an object-based framework for supporting context-sensitive applications. Its context-aware interface definition language (CA-IDL) is used to generate context sensitive objects. These objects run on a customized ORB, which supports communication and context-awareness between the objects. The Gaia [19] project also uses a high level scripting language called LuaOrb [5]. Based on the interpreted language Lua [10], it provides language binding between Lua and CORBA, COM and Java. The project uses the scripting language to program and configure their concept of a pervasive computing environment called *active space*. While also providing the developer with a means of co-ordinating the activities of entities within the active space environment.

Our approach differs from the above in that we look to at using swarm intelligence techniques to facilitate the emergence of pervasive computing environment for collection of independent entities. The novelty of the approach stems from the use of stigmergy and how local contextual information can be used as the *only* communication medium for controlling the behavior of entities in a pervasive computing environment.

8. Summary and future work

We have argued that the use of swarm intelligence techniques, such as stigmergy, can help in the construction of robust pervasive computing environments. Section 2 detailed how the phenomena of stigmergy could be used in these types of environments. Cocoa exploits these techniques to provide a framework which cultivates a self-coordination mechanism for pervasive computing environments. Sections 3 and 4 provided a detail description of the architecture.

Although early results seem promising, there is still substantial further work in validating and improving the Cocoa framework. The range and extent at which the system wide behaviors can emerge from a pervasive environment is still not fully understood. There is a need for a wider range of entities to be developed and deployed so that the mechanisms and implications of using stigmergy can be fully understood. We are currently developing a number of entities and we hope to provide a more complete picture in the near future.

One of our mains concerns at the present time is in the manner that context information is defined. For one entity to understand another entity's situation it must understand the meaning of the context information being sent. Currently there is the concept of primary context information, which is well understood and defined, and secondary context information which consists of key/value pairing that are open to interpretation. To tackle the problem we are in the process of defining an ontology of context information that can be used in Cocoa. Our other concern is the privacy of entities. Even though context information is kept in the local environment, and that entities have full control over the information being propagated privacy may still be a concern and other methods of securing the information may be required. Another concern is that although we provide an attractive programming abstraction for developing pervasive computing environments we have yet to validate the expressiveness of YABS. There is a need to develop range of applications to show that a full complement of applications can be implemented.

Our research into using swarm intelligence techniques in pervasive environments has so far been encouraging and we continue to develop Cocoa and hope to expand the range environments in which it can operate in the coming months.

9. Acknowledgments

The work described in this paper has been partly supported by the Irish Higher Education Authority under the Carmen collaboration between Media Lab Europe and Trinity College. The authors are also grateful to Dublin City Council for supporting the installation of the WAND nodes.

References

[1] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[2] J. F. Allen. Time and time again: the many ways to represent time. *International Journal of Intelligent Systems*, 6:341–355, 1991.

[3] S. A. Brueckner and H. V. D. Parunak. Swarming agents for distributed pattern detection and classification. In *AAMAS*, 2002.

[4] P. Castro and R. Muntz. Managing context data for smart spaces. *IEEE Personal Communications*, 7(5):44–46, October 2000.

[5] R. Cerqueira, C. Cassino, and R. Ierusalimschy. Dynamic component gluing across different componentware systems. In *International Symposium on Distributed Objects and Applications (DOA'99)*, 1999.

[6] A. Dey and G. Abowd. Towards a better understanding of context and context-awareness. In *Workshop on The What, Who, Where, When, and How of Context-Awareness, as part of the 2000 Conference on Human Factors in Computing Systems (CHI 2000)*, April 2000.

[7] A. K. Dey and G. D. Abowd. The context toolkit: Aiding the development of context-aware applications. In *Workshop on Software Engineering for Wearable and Pervasive Computing (CHI '99)*, May 1999.

[8] P.-P. Grassé. Le reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la theorie de la stigmergie: essai d'interpretation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.

[9] O. Holland and C. Melhuish. Stigmergy, self-organization, and sorting in collective robotics. *Artif. Life*, 5(2):173–202, 1999.

[10] R. Ierusalimschy, L. H. de Figueiredo, and W. Celes. Lua-an extensible extension language. *Software: Practice and Experience*, 26(6):635–652, 1996.

[11] I. Kassabalidis, M. El-Sharkawi, R. Marks, P. Arabshahi, and A. Gray. Swarm intelligence for routing in communication networks. In *IEEE Global Telecommunications Conference*, 2001.

[12] C. D. Kidd, R. J. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. Mynatt, T. E. Starner, and W. Newstetter. The aware home: A living laboratory for ubiquitous computing research. In *Second International Workshop on Cooperative Buildings - CoBuild'99*, 1999.

[13] R. Meier and V. Cahill. Steam: Event-based middleware for wireless ad hoc networks. In *International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02)*, pages 639–644, 2002.

[14] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[15] C. Pinhanez and A. Bobick. Fast constraint propagation on specialized allen networks and its application to action recognition and control. MIT Tech Report 456, MIT, January 1998.

[16] C. Pinhanez and A. Bobick. It/i: A theater play featuring an autonomous computer graphics character. In *ACM Multimedia'98 Workshop on Technologies for Interactive Movies*, January 1998.

[17] C. Pinhanez, K. Mase, and A. Bobick. Interval scritps: a design paradigm for story-based interactive systems. In *CHI'97*, March 1997.

[18] C. S. Pinhanez. *Representation and Recognition of Action in Interactive Spaces*. PhD thesis, MIT, June 1999.

[19] M. Roman, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[20] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven, and W. V. de Velde. Advanced interaction in context. In *1th International Symposium on Handheld and Ubiquitous Computing (HUC99)*, pages 89–101. Springer, 1999.

[21] M. Weiser. The computer for the twenty-first century. *Scientific American*, pages 94–104, September 1991.

[22] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. Gupta. Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing*, 1(3):33–40, 2002.