VIRTUAL APPLIANCES FOR PERVASIVE COMPUTING: A DECONSTRUCTIONIST, ONTOLOGY BASED, PROGRAMMING-BY-EXAMPLE APPROACH

J.S.Y. Chin, V. Callaghan, M. Colley, H. Hagras, G. Clarke
Department of Computer Science, University of Essex, UK
Email: iieg@essex.ac.uk Web: http://iieg.essex.ac.uk

ABSTRACT

People are going to experience a revolution in the nature and capability of their home environment in a future where domestic electronic artefacts containing embedded computers and network connections opens up the possibility for hundreds of communicating devices cooperating in communities serving the occupant – this is the "Pervasive Computing" vision. This paper describes research in progress that takes the notion of collaborating home artefacts forward in four ways; (a) it introduces a model that promises to change the nature of home appliances - the deconstructed appliance model (b) it introduces a novel approach to programming pervasive computers called Task Oriented Programming (TOP), (c) it presents a Deconstruction and Community Programming (dComp) ontology supporting the formation and programming of coordinated communities of home appliances and (d) it describes the iDorm a test-bed for this work. We support the theoretical ideas in the paper with details of our implementation and initial evaluation work that shows that TOP operations, such as queries, can be completed in under a second. The work described in this paper is funded by the UK DTI Next Wave and Markets Technology programme.

1.0 INTRODUCTION

1.1 Background

Embedded-computer technology is developing at a breathtaking pace. According to industry statistics, a staggering 98% of the world's production of microprocessors (some 8 billion, in 2001) is integrated into gadgets such as video recorders, washing machines, mobile phones and other embedded-computer based appliances [14]. For example, it is estimated that there are at least 680 million mobile phones in the world [5]. While these technological advances are fuelling significant changes in both the high-tech marketplace and living-environments, the most radical paradigm shift will originate from the way these technologies are applied. Communities of appliances (collectives) will be able to collaborate to provide new synergetic functionalities e.g. the telephone ringing can be made to interact with other devices to carry out other functions, such as pausing the TV, creating higher order "virtual appliances" [3]. The nature of appliances themselves will be questioned. Are appliances monolithic artefacts containing inaccessible sets of fixed functionalities or are they artefacts whose sub-functionalities are visible and accessible to other users and devices and can be logically or physically, deconstructed and reconstructed differently? The role of the end-user changes as tools to design novel functionalities from communicating communities of coordinating devices become available. The main implication arising from this vision is that techniques for describing the knowledge within such systems need to be found e.g. the devices and their capabilities, together with methods that would allow end-users to manage and program communities of coordinating pervasive computing devices, without incurring prohibitive cognitive loading. We argue that the key enabling technologies in realising this vision are an end-user programming tool and a methodology for describing networked device capabilities, particularly a means to describe the notion of community. Thus we have developed an end-user programming tool, TOP, and an ontology, dComp that supports this paradigm, both of which form the main focus, and original contribution, of this paper.

1.2 The Deconstructed Appliance Model

Whilst traditional stand-alone home appliances provide very useful functionality to users, when you add a network connection a number of significant possibilities arise. It becomes possible to access individual sub-functions within the appliance allowing, for example, the on/off switch in a light to be emulated by software on a PC thus enabling remote control of home appliances. More significantly it allows the functional units that make up current appliances to be shared. For example, the audio amplifier in a TV could be used by the HiFi system, or vice versa. Thus, *virtual appliances* could be created by establishing logical connections between the sub-functions of appliances, creating replicas of traditional appliances, or inventing altogether new appliances [3]. In essence this paradigm involves the *deconstruction* of traditional appliances into their atomic functionalities (either physically or logically), allowing the user to re-construct virtual appliances by reconnecting the basic atomic functionalities in various ways, we call this the ***deconstructed appliance*** model. Examples of this approach include SUN's Epsilon Project[1], which is exploring how appliances are decomposed into small independent devices each having a virtual world proxy which can be "connected" to other proxies to create meta systems (offering conventional appliance functions, or novel ones created by the user "invented" combinations). A particularly interesting aspect of the Epsilon work is that it explores the notion of ultra-thin clients where the physical manifestation of the appliance becomes near stateless with most state and process residing on proxies whose location is almost irrelevant. The work at SUN is wide

[1] http://research.sun.com/projects/epsilon/

ranging and includes studies on supporting middleware [19]. As part of their EasyLiving[2] project Microsoft are also exploring the notion of deconstruction (dis-aggregation in their terminology) to PCs and services, demonstrating how a disconnected "pool" of screens, keyboards and applications can be dynamically (and automatically) re-connected to recreate a virtual PC for a user in differing contexts (eg the screen may change from a beamer to a TFT depending on the user's position within a room). Other work at Essex (beyond that described in this paper) is investigating a user metaphor referred to as "creative misuse" which explores how end-users "program" technology in ways manufactures may not have envisaged by deliberately operating systems in ways they were not designed [20]. The key to creating virtual appliances from deconstructed functions is making connections between these functions so that they form a virtual appliance with its own functionality. It becomes a ***community*** of coordinating devices with a new functionality. This concept of community is not limited to deconstructed appliances, but relates to any set of coordinated pervasive services, whatever their functional level. Clearly the richer the pool of (sub-)functions or services, the greater the possible permutations for the user to create new virtual appliances. However user friendly tools for the creation of such communities are required in order to protect the user from the cognitive loading imposed by the complexity of the task.

1.3 The Task Oriented Programming approach

A critical aspect of this vision is providing the non-technical lay end-users with a means to "program" the coordinated actions of communities of communicating pervasive computing devices. There are numerous approaches to this challenge such as implicit methods that use autonomous intelligent agents [10] [7] that aim to reduce the cognitive load on the user aspects or explicit methods that use end-user programming approaches [15] [12] which seek to introduce creativity and transparency. an be traced to work such as Programming-By-Example (PBE) (sometimes referred to as Programming-By-Demonstration - PBD), an end user programming paradigm pioneered by Smith in the mid-seventies [15], Tangible Computing, a way of bringing a physical metaphor to software abstractions pioneered by Ullmer and Ishii [Ullmer 00], Palpable Computing[3] an approach to promote user control and choice through increased visibility of pervasive computing technology, Learning-From-the User (LFU), an embedded-agent learning paradigm Essex University has been developing for many years [3] and ontology mainly drawn from research work on the semantic web [1]. The underlying principle is that a person sets a community into a "teaching mode" and then demonstrates the behaviour required from the system, by either physically, or graphically, using the system.

1.3 Programming-By-Example

Programming-by-example (PBE) was introduced by Smith in the mid-seventies, where the algorithms for the system functionalities were not described abstractly but rather demonstrated in concrete examples [15]. Henry Lieberman later described PBE is "*a software agent that records the interactions between the user and a conventional direct-manipulation interface and writes a program that corresponds to the user's actions*", where "*the agent can then generalise the program so that it can work in other simulations similar to, but not necessarily exactly the same as, the example on which it is taught*" [12]. Thus, PBE reduces the gap between the user interactions and the delivered program functionality by merging the two tasks. The main area of PBE work has focused on graphical user interfaces running on PCs. By way of a few examples, PBE has been applied to computer application development; Computer-Aided Design (CAD) system, children's programs, World Wide Web related technologies and home automation, [2], [8],[13],[12]. The underlying principles in PBE are generic and transportable to the pervasive computing world. In addition to the underlying scientific principle PBE shares the same motivation of empowering lay-end-users to utilise what would otherwise be prohibitively complex technology. However, to-date PBE has not been applied to programming tangible physical objects, nor any other aspect of pervasive computing. Thus TOP is the first application of PBE to pervasive computing. Further details on the TOP paradigm and progress in achieving it are presented in section 3.

1.4 The dComp Ontology

One similarity between all the projects that address pervasive computing is the need to encapsulate environmental information. The lack of a standard way of describing such knowledge is seen as an obstacle for independently developed software applications to interoperate [4]. One standard approach is an ontology which defines the terms used to describe and represent an area of knowledge. This consists of sets of well-defined vocabularies and associated semantics that can be reasoned about. Although XML, DTDs and XML Schemas are sufficient for exchanging data between parties who have agreed to the definitions beforehand, their lack of semantics make reasoning (or even merging information) across diverse communities difficult.. Until recently, most ontology work focused on the Semantic Web. Many well-defined ontologies have been developed. For example, in bioscience, the GENE Ontology and the MGED ontology in the human domain, the "Food Ontology" and "Wine Ontology", were developed by the W3C and in the Information Science domain, the SUMO ontology has been flourishing for promoting information processing related activities. However, the most relevant ontology standard to dComp is SOUPA, developed by UbiComp which defines a set of generic vocabularies for ubiquitous and pervasive applications [4]. Although SOUPA promotes the interoperability of information between applications, it is based on a context-awareness model, and as a result, it is particularly well suited in the context-awareness domain. This model differs from dComp, in that our primary focus is on *coordinating* actions of communities of home based appliances, and deconstructed functions to produce higher level community functionality. This concept has not been fully addressed by the current SOUPA standard. In addition,

[2] http://research.microsoft.com/easyliving/
[3] http://www.ist-palcom.org/

SOUPA provides only limited support for the concept of home UPnP-based devices. In order to realise our vision, a set of explicitly well-defined vocabularies (i.e. an ontology) is needed to model, not just to describe the basic concept of deconstructed devices (or to deal with UPnP) but also, the communities they form, the services they provide, the rules and policies they follow, the resultant actions that they take, and of course the people who inhabit the environment along with their individual preferences. Thus, we have built on the SOUPA work by defining vocabularies to provide direct support for community coordination, deconstructed appliances and home environments.

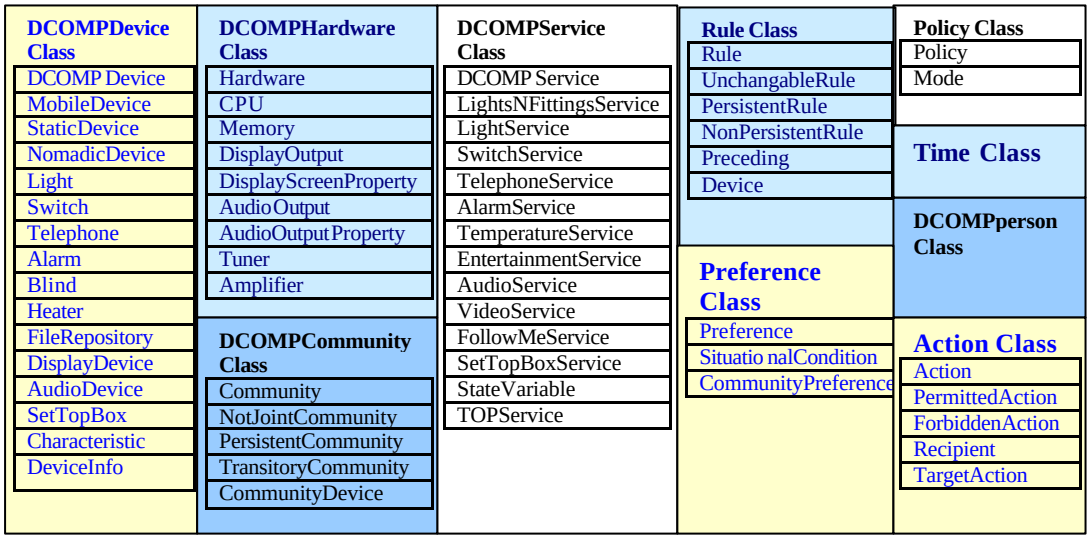

Figure 1 - dComp ontology (v.1.1)

We have chosen to model the dComp ontology using OWL, the Semantic Web standard Language developed by the semantic web initiative sponsored by the World Wide Web Consortium (W3C). OWL provides a framework for asset management, in particular it facilitates greater machine interoperability for sharing and reusing of web data than is supported by structured data markup languages such as eXtensible Markup Language (XML), Resource Description Framework (RDF) and RDF Schema (RDF-S). RDF is the syntax generally used in the Semantic Web for representing data. The data structure of RDF is in the form of triples[2], and each triple is referred as a resource which uniquely identified by a URI. OWL shares the same root as its predecessor, DAML-OIL, both use RDF as their main modelling language to define vocabularies together with XML as the syntax for representing information. We chose OWL for three main reasons. Firstly, the language is now a standard with the backing of a well known and highly regarded standards organisation. Secondly, the language is much more expressive than RDF or RDF-S by providing additional formal semantic vocabularies allowing us to embed more information into our ontology. Thirdly, it enjoys widespread support from developers of Semantic Web tools such as Jena[4] [19] and inference engines such as RACER [9] and F-OWL [18]. More information on the dComp ontology and current progress in implementing is given in section 4.

[2] An **RDF triple** contains three components: (1) the **subject**, which is an RDF URI reference or a blank node (2) the **predicate**, which is an RDF URI reference (3) the **object**, which is an RDF URI reference, a literal or a blank node

[4] http://jena.sourceforge.net/

2.0 THE IDORM TEST ENVIRONMENT

For our experimental work on TOP we are using a pervasive computing test-bed called the iDorm at the University of Essex, which takes the form of a student bed-sitting room, see figure 2. The iDorm is a multi-use, multi-user space containing areas for different activities such as sleep, work and entertaining. It comprises approximately 30 networked functions built into devices such as telephones ,MP3 players, lights, beds and chairs. Connectivity and a common interface to the iDorm devices are implemented via IP networking and Universal Plug and Play (UPnP). UPnP is a distributed middleware that employs event-based communication and supports automatic discovery and configuration. Our experimental TOP architecture makes extensive use of UPnP as its underlying network communication infrastructure.

3.0 TOP

3.1 TOP Overview

The motivation behind TOP was to create a system that maximised user control and transparency whilst minimising the need for detailed technical knowledge. This was driven by experience with autonomous agent based systems where some users expressed fears related to privacy - not knowing what the agents were doing, when they were doing it and who or what was accessing the information. [6] [21]. In the TOP approach, the system is *explicitly* put into a "learning" mode and is taught (by demonstration) how to behave by the lay end-user. Virtual appliances could be created by establishing logical connections between the sub-functions of appliances,

creating replicas of traditional appliances, or inventing altogether new appliances.

Figure 2 – The iDorm

Users that don't want to program new functions, are free to use the stand-alone appliance. In addition, the vision for TOP includes the notion of *pre-fabricated interconnection templates* which are descriptions of pre-made communities, such as a TV. The key to creating such virtual appliances is that of making connections between network functions to form a community of coordinating services. To facilitate this it is necessary to have some standard way of describing the functionality of the devices and connections. For TOP, we utilise the dComp ontol (see next section for details). Clearly, this concept of *community* relates to any set of coordinated pervasive entities, whatever their functional or physical level (i.e. ultimately, it could relate to any level of granularity that had appropriate components and communications from nano to macro scale building to building, city to city environments). In general, the richer the pool of (sub-)functions, the greater the number of possible virtual appliances.

.

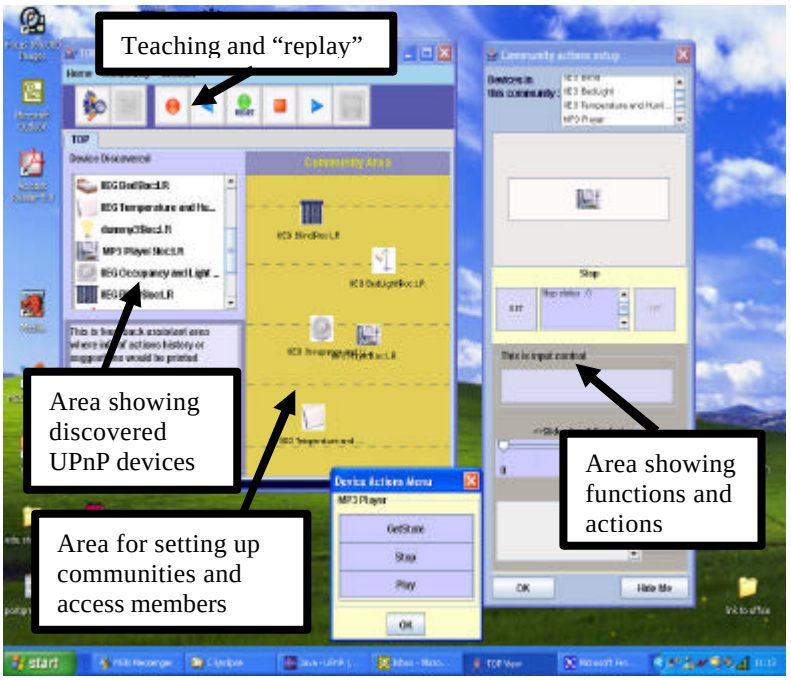

Figure 3 – The TOP Editor with example community set up

2.2 TOP Architecture

As TOP has the notion of working with communities the system supports setting up communities (sets of communicating devices). By selecting a community that the user wishes to program, a set of coordinated actions are taught to the system by simply using the home networked devices in a role-play mode, supported by some on-screen activities An action causes an appliance to generate an associated event, and this event is then used to generate appropriate rules (based on a snapshot of the environment). More generally, coordinating actions (i.e. tasks) are performed by a community (i.e. one or more devices). A device can be involved in more than one community (i.e. performing one or more actions). The user interface with TOP is via an editor called TOPeditor (see figure 3). This editor provides a means for: (1) displaying discovered devices, (2) setting up / amending communities and (3) managing learning sessions. Tasks can be taught via interacting with on-screen or physical devices. The TOP architecture, shown in Figure 4, comprises the following main modules:

- TOPeditor – the user interface used to program and interact with the system.
- TOP Engine – responsible for discovering and subscribing to community events and contains a

Rule Manager that is responsible for gathering, generating and executing rules, together with an Eventing Handler that manages TOP events.

- Data Modelling Manager – responsible for maintaining and providing consistent data.
- Community Manager - manages the communities of coordinating devices.

To facilitate the information to be used within and beyond the community, data needs to be standardised so that it can be understood by all other parties in the network. For this aspect of work, the semantics in the dComp ontology supports information interoperability between applications, providing a common machine "understanding" knowledge framework.

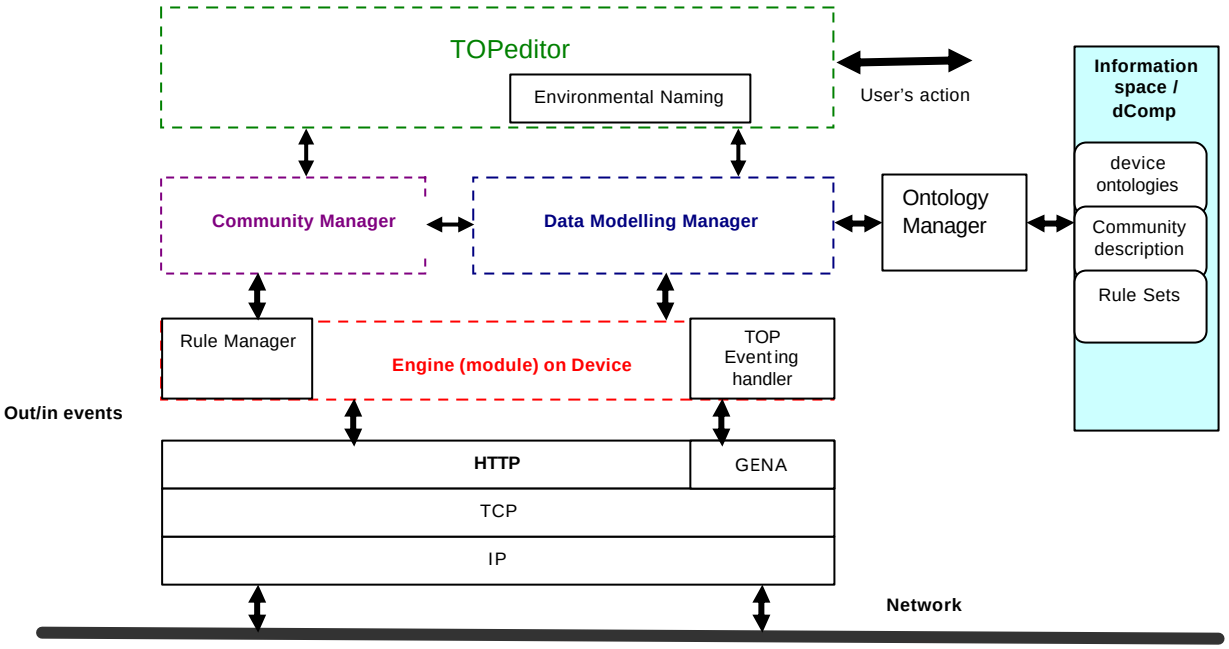

Figure 4 - The TOP Architecture

3.3 TOP Progress to Date

Using the iDorm we have been able to validate the basic components of the TOP architecture. In addition to the iDorm infrastructure, we have also augmented our experimental capability by creating a number of simulated UPnP devices including telephones, lights and MP3 players. We have implemented the TOP Engine component, which is responsible for interfacing with the UpnP network as well as managing the task of sending and receiving events. We have implemented the Data Modelling Manager component, which is responsible for translating UpnP parameters to the TOP internal data structure which is based on the dComp ontology model thereby providing the system components with a set of consistent data. We have completed the Community Manager, which is responsible for tasks such as setting up of communities, checking and keeping track of what devices have joined or left the community and subsequently informing other interests parties about these events. (A secondary function of the community manager is to manage the device mode, during user interaction.) The TOPeditor is the interface component allowing the user to interact with the system (user can also interact physical devices after setting the device mode within the TOP session). It is responsible for presenting information to the users on communities and their properties together with providing a means for the user to interact and manipulate them. TOPeditor has been constructed with a dynamically configured control panel that provides the means for the user to directly control and interact with the networked devices. The TOPeditor also contains a history panel that retains a record of the user's activities. Three rule panels' present rule related information. An Environmental Naming module is provided to give devices meaningful names. The current TOP naming convention used by this module is to seek to base names on location (current location is predefined). We have completed and tested all of the above components on a live UpnP device network. Currently work is underway to complete the last module, the Rule Manager which should be completed well ahead of the workshop.

4.0 THE DCOMP ONTOLOGY

4.1 Introduction

In this section, we describe the key ontology concepts in the current version (v.1.1) of dComp. The full version is available on the web at http://iieg.essex.ac.uk/dcomp/ont/dev/2004/05/. To avoid any confusion, henceforth we refer to the dComp Ontology as dComp whereas the documents that describe entities (e.g. device, services, community etc) that exist in the dComp environment are referred to as documents. Figure 1 shows a complete list of names of classes in the current version of the dComp ontology. dComp also describes a set

properties and relationships (that are associated with these concepts, along with the restrictions they may have.

4.2 Namespace

The base namespace for dComp is given at http://iieg.essex.ac.uk/dcomp/ont/dev/2004/05/. However, for ease of file management, every dComp document was built under this base namespace plus a name associated with its main class. For example, the DCOMPDevice document namespace is: http://iieg.essex.ac.uk/dcomp/ont/dev/2004/05/DCOMPDevice#, and the DCOMPCommunity document namespace is http://iieg.essex.ac.uk/dcomp/ont/dev/2004/05/DCOMPCommunity# and so forth.

4.3 DCOMPDevice

UPnP™ technology is perhaps the leading technology to enable simple and robust peer-to-peer connectivity among devices and PCs. Our pervasive computing test-bed, the iDorm (has been built on top of UPnP, and, our first model of deconstructed pervasive devices was based on UPnP technologies. The notion of dComp Devices refers to networked devices, or home appliances, where the functionalities are similar to those we would normally find in a home. The main class called DCOMPDevice provides a generic description of any devices/appliances. Another class called DeviceInfo is for all individuals which share some UPnP descriptions. Because of problems relating to UPnP vulnerabilities it will probably take some time for *all* home electronic devices/appliances to become UPnP compatible.

```
<device:DisplayDevice rdf:ID="CRT monitor">
 <rdfs:label xml:lang="en">CRT monitor</rdfs:label>
 <device:hasDeviceInfo>
  <device:friendlyName>Philips 17 CRT monitor</device:friendlyName>
  <device:DeviceUUID>UUID:PHLCRT17</device:DeviceUUID>
  <device:DeviceType>urn:schemas-upnp-
org:Philips17CRTmonitor:1</device:DeviceType>
<device:DeviceModelURL>http://iieg.essex.ac.uk/dComp/onto/Philips17CRTmonitor/</d
evice:DeviceModelURL>
  <device:DeviceModelNumber>107T61/05</device:DeviceModelNumber>
</device:hasDeviceInfo>
<hw:componentOf>
 <hw:DisplayOutput rdf:ID="17CRT"/>
  <hw:hasDisplayScreenProperty>
   <hw:width rdf:datatype="&xsd;int">1024</hw:width>
   <hw:height rdf:datatype="&xsd;int">768</hw:height>
  </hw:hasDisplayScreenProperty>
</hw:componentOf>
<device:hasCharacteristic rdf:resource="#Nomadic"/>
 <serv:hasDCOMPService>
  <serv:DisplayService>
   <serv:serviceID>urn:iieg-essex-ac-uk:serviceId:DisplayCRT</serv:serviceID>
    <serv:hasStateVariable>
       <serv:name>DisplayCRT</serv:name>
        <serv:value rdf:datatype="&xsd;int">0</serv:value>
       <serv:evented rdf:datatype="&xsd;boolean">true</serv:evented>
    </serv:hasStateVariable>
  </serv:DisplayService>
 </serv:hasDCOMPService>
</device:DisplayDevice>
```

Figure 5 - Typical DisplayDevice Expression

For these reasons, the DeviceInfo class has only partial restrictions to UPnP related properties; the classes are: (1) deviceUUID (2) friendlyName (3) deviceType (4) deviceModelURL. The dComp environment, supports devices with different mobility characteristics. To model these, we have defined a class called Characteristic that generalise these via use of one of these classes: StaticDevice, NomadicDevice or MobileDevice. As dComp defines a *closed-world* then any device/appliance is also a DCOMPDevice (i.e. By OWL equivalentClass). The DCOMPDevice includes both nuclear (traditional appliances) and atomic (deconstructed) devices. Currently DCOMPDevice has 10 sub-classes, these are: Light, Switch, Telephone, Alarm, Blind, Heater, FileRepository, DisplayDevice, AudioDevice, SetTopBox. The DCOMPDevice class (including its sub-classes) have associated containment relationships. These relationships are defined by using OWL object property[5] and they are: (1) hasDeviceInfo (2) hasHardwareProperty (3) hasDCOMPService (4) hasCharacteristic. A typical DCOMPDevice can be expressed as shown in figure 5.

4.4 DCOMPHardware

We define an abstract class, DCOMPHardware, that generalises all hardware that exists in DCOMPDevice. DCOMPHardware has 8 sub-classes along with associated properties. They are: CPU, Memory, DisplayOutput, DisplayInput, AudioOutput, AudioInput, Amplifier and Tuner. The CPU class has two properties: speed and speedUnit. To model the relationship between a DCOMPDevice and its hardware, we define a SymmetricProperty[6] "componentOf" that links the DCOMPDevice to the range of DCOMPHardware. With this SymmetricProperty, we could express:

```
<hw:CPU rdf:ID="IntelIPX255">
   <hw:speed
rdf:datatype="&xsd;int">400</hw:speed>
   <hw:speedUnit
rdf:datatype="&xsd;string">MHZ</hw:spee
d>
</hw:CPU>
<device:MobileDevice
rdf:ID="JCpocketPC">
   <componentOf
rdf:resource="#IntelIPX255" />
</device:MobileDevice >
```

The statement show's there is a mobileDevice called JCpocketPC, and it is the componentOF a CPU called "IntelPXA255"whose speed is 400MHZ, and vice versa. Memory Class has two properties. They are: amountOfmemory and unitOfMemory. Display Device in a dComp environment can have different resolutions (eg. in the above, the display screen resolution of JCpocketPC is smaller than those LCD screens). Likewise, audio sources demand differing bandwidth depending on the manufacturer. Therefore, two extra classes: DisplayScreenProperty and AudioInOutput Property have been defined to express these needs. Finally, the relationships between the DisplayScreenProperty and the DisplayInput class are linked by the object property "has DisplayScreenProperty" while the relationship between the AudioInOutputProperty and the AudioOutput/Input class are linked by the object properties "has AudioInOutputProperty" respectively.

[5] object property denotes relations between instances of two classes. See owl:ObjectProperty

[6] SymmetricProperty denotes: If a property, P, is tagged as symmetric then for any x and y: P(x,y) iff P(y,x)

4.5 DCOMPService

In order for the DCOMPDevices to work to together, every DCOMPDevice on the dComp network offers services. We model these services by introducing a class called DCOMPService, representing all services on the dComp network. DCOMPService has three sub-classes, namely; TOPService, LightsAndFittingsService and EntertainmentService. The TOPService class is given as an example of a programming-by-example service. Programming-by-example systems are generally composed of two services, an execution engine and an editor. Thus in this example TOPService is described as a collection of TOPEngineService and TOPEditorService. The editing service will have various sub-services. Thus, TOPEditorService includes: EditingService, SettingUp CommunityService, and ConfigurationService. The LightsAndFittings Service class denotes a set of Lights and Fittings services include: Light Service, Telephone Service, Alarm Service, Heater Service, and Temperature Service. The EntertainmentService class corresponds to the services include: AudioService, VideoService, FileRepositoryService, SetTopBoxService and FollowMeService. A DisplayService, which is the same as VideoService is also introduced. TOPService, LightsAndFittingsService and EntertainmentService are mutually distinct, thus we model this characteristic by declaring these classes to be disjointWith[7] each other. Every dComp service is identified by its service ID. Thus every DCOMPService has a property called "serviceID". We also define a class called "StateVariable" to represent UPnP values. The StateVariable class has three properties, namely: "name", "value" and "evented". The relationship between a DCOMPService and the StateVariable is linked by an object property called "hasStateVariable". The relationship between a DCOMPDevice and DCOMPService is coupled by an object property called: hasDCOMPService.

4.6 Community

Devices in a dComp environment are expected to work as a community (collective). We introduce a class called DCOMPCommunity to represent all communities that exist in the dComp environment. There are three types of communities being modelled namely: (1) SoloCommunity - for those devices that have not been invited to join a community (perhaps newly joined) (2) PersistentCommunity - for those communities that have a degree of permanency (3) TransitoryCommunity - for communities which have a short lifetime.

A DCOMPDevice can join one or more communities. To model the relationship between a DCOMPDevice and a DCOMPCommunity, we define an object TransitiveProperty[8] called "inTheCommunityOf". A class called CommunityDevice is introduced to represent all the devices in a community. These devices are identified by their deviceUUID identification. The relationship between a Community and CommunityDevice is linked by another object TransitiveProperty called "hasCommunityDevice". Of course, a community must have at least one CommunityDevice. This restriction holds for all communities. Communities in dComp are formed and owned by a user. The properties of Communities are: community ID, communityName, communityDescription and timestamp. The relationship between a community and its owner is linked by an object type property, called "hasOwner". A DCOMPTV community can be specified as shown in figure 6.

```
<com:TransitoryCommunity rdf:ID="JCTV">
 <com:communityID>Tran-JCTV</com:communityID>
 <com:communityName>JC TV</com:communityName>
 <com:communityDescription>The first JC testing
TV</com:communityDescription>
 <com:timeStamp rdf:datatype="&xsd;dateTime">2004-09-
06T19:43:08+01:00</com:timeStamp>
 <com:hasOwner>
  <person:Person>
   <person:firstName rdf:datatype="&xsd;String">Jeannette</person:firstName>
   <person:nickname rdf:datatype="&xsd;String">JC</person:nickname>
   <person:gender rdf:resource="#Female"/>
  </person:Person>
 </com:hasOwner>
<com:hasCommunityDevice>
 <com:CommunityDevice>
  <device:deviceUUID>UUID:PHLCRT17</device:deviceUUID>
 </com:CommunityDevice>
 <com:CommunityDevice>
  <device:deviceUUID>UUID:PHLAudioMMS223</device:deviceUUID>
 </com:CommunityDevice>
 <com:CommunityDevice>
  <device:deviceUUID>UUID:NetGem442</device:deviceUUID>
 </com:CommunityDevice>
</com:hasCommunityDevice>
</com:TransitoryCommunity>
```

Figure 6 - Partial definition of a TV community

4.7 Rules

Rules are set for a community for executing coordinating actions. We define an abstract class called Rules to represent a set of rules for a community. We model three types of rules: (1) FixedRules – for those rules that can not be changed, (2) PersistentRules – for those rules that are seldom changed as they are repeatedly executed and (3) NonPersistentRules – for rules that frequently change so only need to execute a few times. Because these three types of rules are mutually distinct, we declare them to be complementOF[9] each other. Every rule has properties: ruleID and ruleDescription. Because rules are set by users, each rule is also has an object property called "hasRuleOwner" to link to the owner. (Note: the rule and community owners may be different people). We define a class called Preceding to represent a set of triggers that cause the coordinating actions to be executed. The devices in the Preceding class are

[7] disjointWith asserts that the class extensions of the two class descriptions involved have no individuals in common.

[8] TransitiveProperty denotes if a device X is in the community of C and the community C is a member of Community P then the device X is also a member of community P

[9] complementOf denotes all individuals from the domain of discourse that do not belong to a certain class

identified by their deviceUUID, and the service they offer. Finally an object property called "hasAction" binds the relationship between Rules and Actions.

```
<NonPersistentRules rdf:ID="Rule1">
 <rule:ruleID rdf:datatype="&xsd;int">00001</rule:ruleID>
 <rule:ruleDescription>Test Rule 1</rule:ruleDescription>
 <com:communityID>Tran-JCTV</com:communityID>
  <rule:hasRuleOwner>
    <person:Person>
    <person:firstName>Jeannette</person:firstName>
    <person:nickname>JC</person:nickname>
    <person:gender rdf:resource="#Female"/>
   </person:Person>
  </rule:hasRuleOwner>
<rule:hasPreceding>
  <!-- can have more than 1 device -->
  <rule:Device>
   <dComp:DeviceUUID>uuid:Telephone01</dComp:DeviceUUID>
   <serv:hasDCOMPService>
   <!-- a device can provide more than 1 service -->
    <serv:TelephoneService>
     <serv:serviceID>Telephone</serv:serviceID>
      <serv:hasStateVariable>
              <!-- a service can have more than 1 value of state variable-->
                <serv:name>state variable 1</serv:name>
                <serv:value>RINGING</serv:value>
     </serv:hasStateVariable>
    </serv:TelephoneService>
   </serv:hasDCOMPService>
  </rule:Device>
</rule:hasPreceding>
<rule:hasAction>
 <act:PermittedAction>
  <act:actionName>Test action</act:actionName>
   <act:hasRecipient>
```

Figure 7 – Example of Rule Structure

4.8 Action

Our Action ontology document has been influenced by the SOUPA Action ontology. We define a class called Action to represent the set of actions that are defined by the rules. As with SOUPA, we have two types of actions, namely: PermittedAction class and ForbiddenAction class respectively. The Action class in dComp is the union of these two action classes. In dComp, every coordinating action has its target devices. To model these target devices, we define a class called Recipient, that represents a set of target devices where actions take place. The members of Recipient are identified by their deviceUUID and serviceID. We call the actions for the Recipient TargetAction. In the TargetAction class, we define two properties namely: actionName (name of the action) and targetValue. A typical statement "when the phone rings, mute the TV" could be expressed as in Figure 7.

4.9 DCOMPperson, Policy and Time

Wherever possible we have sought to build on existing ontology work. SOUPA provides a suitable DCOMPperson, Policy and Time ontology and thus these have been adopted in dComp. Due to space restrictions, we are not explaining them here; for further information refer to their site at: http://pervasive.semanticweb.org/soupa-2004-06.html

4.10 DCOMPPreference

As the name implies, DCOMPPreference describes the preferences a person has within any given set of options. In dComp, preferences are referred as "situated preferences", similar in idea to Vastenburg's "situated profile" where he uses situations as a framework for user profile so that the values of the profile are relative to situations [17].

```
<owl:Class rdf:ID="SituationalCondition">
  <rdfs:label>SituationalCondition</rdfs:label>
  </owl:Class>
  <SituationalCondition rdf:ID="DuringTheWorkdays"/>
  <SituationalCondition rdf:ID="DuringTheWeekends"/>
  <SituationalCondition rdf:ID="WhileOutOfTown"/>
  <SituationalCondition rdf:ID="WorkingFromHome"/>
  <SituationalCondition rdf:ID="FriendsVisiting"/>
  <SituationalCondition rdf:ID="FamilyVisiting"/>
  <SituationalCondition rdf:ID="OnHoliday"/>
  <SituationalCondition rdf:ID="WhenComeHomeFromWork"/>
  <SituationalCondition rdf:ID="WhenComeHomeFromSchool"/>
  <SituationalCondition rdf:ID="WhenAtMyOffice"/>
  <SituationalCondition rdf:ID="WhenDining"/>
  <SituationalCondition rdf :ID="WhenHavingLunch"/>
  <SituationalCondition rdf:ID="WhenHavingBreakfast"/>
  <SituationalCondition rdf:ID="WhenEating"/>
  <SituationalCondition rdf:ID="WhenPlayingComputerGames"/>
  <SituationalCondition rdf:ID="WhenWatchingTV"/>
  <SituationalCondition rdf:ID="AtNight"/>
  <SituationalCondition rdf:ID="InTheMorning"/>
  <SituationalCondition rdf:ID="AtLunchTime"/>
  <SituationalCondition rdf:ID="AtTeaTime"/>
  <SituationalCondition rdf:ID="Alone"/>
  <SituationalCondition rdf:ID="WhenAlarmGoesOff"/>
  <SituationalCondition rdf:ID="WhenSmokeAlarmGoesOff"/>
```

Figure 8 – Example of Context Condition

We define a class called Preference to represent a set of contextual preferences of a person for his community. This Preference class has a subclass called CommunityPreference and an associated property called communityID. To model "person A prefers X, depending on the context of Y", we define another class called SituatedConditions. This class represents the set of contextual conditions which the person's preferences depend on. Although a person is allowed to define his own SituatedConditions, we also explicitly define a list of pre-set situated conditions so that it forms a default template that a person can use. Figure 8 shows the default template of context conditions. The Preference class has a close relationship to the Person class. To bind this relationship, we define an object property called "hasPreference" which links the domain of Person to the range of Preference. The relationship between the Preference class and SituationConditions class is linked by another object property called: "hasCondition".

4.11 dComp Progress to Date

The biggest concern about the use of ontology in pervasive systems is processing speed, especially for functions such as device queries. In our iDorm architecture, we employ a distributed hierarchical system in which the power of computer units range from minimal processors (e.g. 20Mhz, 1MB) through to maximal embedded systems (e.g. 4Ghz, 100MB) allowing a full range of granularity distribution experimentation. We envisage there being a mixture of processors, some being powerful enough to support UPnP stacks., agents or TOP, others, perhaps handling simple sensors, not being so able and therefore requiring proxies. In our tests we found the performance of dComp running on a 2Ghz processor and executing a query on 32 devices (which is the current limit of our test-bed) was of the order 600ms

which, if this was part of a user interaction process, is just within what might be acceptable. Therefore we envisage needing the more powerful processors to support the TOPengine.

5.0 SUMMARY

This paper has reported on research in progress within a deconstructed appliance domain towards the development of an end-user programming architecture (Task-Oriented-Programming) for pervasive computing. This includes a run-time environment (TOPengine) and an end-user tool (TOPeditor) that enables lay-users to program coordinated actions in groups of networked pervasive computing devices. We explained that this deconstructed model is radical departure from the current situation where appliance functionality is designed and fixed by the manufacturer. This approach is motivated by our belief that creativity, privacy and transparency (i.e. understanding and trust) are essential issues to users of pervasive network based appliances. In pursuit of this vision we described components we have built and tested to date namely the TOP system and supporting ontology dComp. TOP is the only interpretation of programming-by-example to be directed at pervasive computing. dComp distinguishes itself from other ontologies in that it directly supports the concept of community, and collectives of devices coordinating actions to create meta-group functionalities. The TOP approach involves the user explicitly in the learning phase, making the system transparent to her. The user plays a critical role in TOP and hence, in addition to continuing the technical development work, our future plans include a significant user trial to evaluate how successful it is in fulfilling its goals of enabling non-technical lay users to program coordinated actions in pervasive computing systems. We look forward to reporting on our progress in this ongoing work and demonstrating our achievements at the workshop.

Acknowledgements
We are pleased to acknowledge the DTI's New Wave Technologies and Markets programme which, via the PHEN project, supported aspects of the TOP research and University of Essex which facilitated the ontology related research as part of Jeannette Chin's personal PhD. In addition we wish to record our gratitude to Phil Bull, Rowan Limb and Alex Loffler of BT's Pervasive ICT Research Centre and to Roy Kalawsky & Scott Armitage of Loughborough University's Centre for the Integrated Home for their part in enabling PHEN to be such an interesting research project.

REFERENCES:

[1] Berners-Lee T, Hendler J, Lassila O "*The Semantic Web*", Scientific American, May 2001

[2] Alan F. Blackwell, Rob Hague, *"Designing a Program Language for Home Automation"*, in G.Kadoda (Ed). Proc PP1G 2001 pp85-103.

[3] Callaghan V, Colley M, Hagras H Chin J, Doctor F, Clarke G "*Programming iSpaces: A Tale of Two Paradigms*" Chapter 24, Springer-Verlag, July 2005

[4] Chen *H* Finin T, and Joshi*l A., "SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications*", Int'l Conf on Mobile and Ubiquitous Systems: Networking & Services, August 2004.

[5] Chin JSY, Callaghan V, *"Embedded-Internet Devices: A Means Of Realizing The Pervasive Computing Vision*", IADIS Int'l Conf, Algarve, Portugal, 5-8 Nov 2003

[6] Chin J, Callaghan V, Clarke G *"Pervasive Information Systems: Issues for the Individual & Society"*, Pervasive Information Systems, M.E.Sharpe New York, Aug 05

[7] Doctor, F., Hagras,H.A.K., Callaghan,V., 'An Intelligent Fuzzy Agent Approach to Realising Ambient Intelligence in Intelligent Inhabited Environments', IEEE Transactions on Systems, Man & Cybernetics, Part A: Systems & Humans, 2004

[8] Guibert N. Girard P., "*Teaching and Learning Programming with a Programming by Example System*", International Symposium on End User Development, Sankt Augustin (Bonn), Germany, EUD - net, 2003

[9] V.Haarslev and R. Moller, Description of the RACER system and its application, In proceedings International Workshop on Description Logics (DL-2001), 2001.

[10] Hagras, H. A. K., Callaghan, V., Colley, M. J., Clarke, G. S., and Duman, H., "A Fuzzy Logic Based Embedded Agent Approach to Ambient Intelligence in Ubiquitous Computing Environments, IEEE J. Intelligent Sys 2004

[11] McBride B.,"*Jena: A Semantic Web Toolkit*", IEEE Internet Computing Nov./Dec. issue 2002

[12] Lieberman H, "*Your wish is my command*", Morgan Kaufmann Press, 2001.

[13] Myers B.A., "*Creating user interfaces using programming by example, visual programming, and constraints*", ACM Transactions on Programming Languages & Systems., 12:2, April 1990, pp 143 – 177

[14] Metcalfe R, *"Keynote Presentation*", ACM1 Conference, San Jose Convention Centre, California, 12-14 March 01

[15] Smith, D. C., "*Pygmalion: A Computer Program to Model and Stimulate Creative Thought"*, Basel, Stuttgart, Birkhauser Verlag. 1977.

[16] Sugiura A,Koseki Y. " *Internet scrapbook: automating Web browsing tasks by demonstration"* Proc 11th annual ACM symp on User interface software & technology, San Francisco 1998, pp.9-18.

[17] Vastenburg M, "*SitMod: a tool for modelling and communicating situations*", Pervasive 2004, Vienna Austria, April 21-23, 2004, ISBN: 3-540-21835-1

[18] Young Zou, Harry Chen, and Tim Finin, *"F-OWL: an Inference Engine for Semantic Web"*, Proceedings of the Third NASA-Goddard/IEEE Workshop on Formal Approaches to Agent-Based Systems, 26 April 2004.

[19] Horan B "*The Use of Capability Descriptions in a Wireless Transducer Network"*, Sun Microsystems Laboratories Report Number: TR-2005-131,Feb 1, 2005

[20] DiDuca D, Van Helvert J "*User Experience of Intelligent Buildings; A user-Centered Research Framework"*, Intelligent Environments 2005, Colchester, 28-29th June 2005

[21] Basu J, Callaghan V "Towards A Trust Based Approach To Security And User Confidence In Pervasive Computing Systems", IE05, Essex, 28-29th June 05