Feng Wang and Kenneth J. Turner. Towards Personalised Home Care Systems.
In Ilias Maglogiannis, editor, Proc. 1st International Conference on
Pervasive Technologies related to Assistive Environments,
pages L2.1-L2.7, Association for Computing Machinery, New York, USA, July 2008.

Towards Personalised Home Care Systems

Feng Wang
Computing Science and Mathematics
University Of Stirling
Stirling, FK9 4LA
Scotland
+44 1786 467428
fw@cs.stir.ac.uk

Kenneth J. Turner
Computing Science and Mathematics
University Of Stirling
Stirling, FK9 4LA
Scotland
+44 1786 467423
kjt@cs.stir.ac.uk

ABSTRACT
Home care is increasingly seen as a promising alternative to traditional care services. Programming home care systems remains a significant challenge considering the potentially large scale of deployment, the differences between individual care needs, and the progressive nature of ageing. In this paper, we present ongoing work on programming home care systems to support personalisation, adaptability over time, and dependability. A policy-based approach is used to build such systems. We present the technical details of our approach, including a policy language for home care and the corresponding system architecture. Policy examples are used to illustrate how the approach supports personalisation of home care services.

Categories and Subject Descriptors
D.1 [**Programming Techniques**]; C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks - Distributed Systems

General Terms
Management, Design

Keywords
home care, policy-based management, personalisation, pervasive computing

1. INTRODUCTION

The growing number of older people is creating more pressure on the resources of existing care services. Increasingly, providing care at home is seen as a promising alternative to traditional healthcare solutions. By making use of sensors, home networks and communications, older people can be helped to prolong independent living in their own homes. Remaining in a familiar environment while being cared for improves their quality of life. Their families and informal carers can also be relieved of worry as to whether those in care are well.

The authors are part of a multi-partner team working on the MATCH project (Mobilising Advanced Technologies for Care at Home, *www.match-project.org.uk*). Through discussion with our social care partners, healthcare practitioners, and other researchers in this area, we have identified the following requirements for home care systems.

Personalisation Each user has unique circumstances and needs. Users vary in their health. For example, older people are prone to diseases such as hypertension, diabetes and Alzheimer's. They may also suffer from functional disabilities such as impaired mobility, falls and memory loss. Users also vary in their personal wishes, including lifestyle and privacy preferences. The combination of these factors varies from person to person. As a result, potential risks can be different for each user receiving care at home. This entails different needs for support and monitoring, as well as requirements for home care products and services.

Customisation Since home care systems may be deployed on a large scale, personalisation should not have to be undertaken by specially trained system integrators and developers. Rather, care professionals should also be able to personalise the system as they have the best knowledge of the care situation.

Adaptation Ageing is a progressive process. As time goes by, new symptoms may appear and existing symptoms may further develop. Home care systems must therefore support easy modification to existing care services, as well as deployment of new services to tackle these changes. Both hardware and software changes may be required.

Dependability Since home care systems directly affect health, welfare and safety, care services must operate in a predictable and dependable way. In general, dependability may involve both hardware and software, though this paper focuses mainly on software aspects. A care service must not exhibit undesirable behaviour, and there should not be interference among care services.

Many research prototypes of home care systems have been created in an *ad hoc* way and require specialised programming. The prototypes are usually built to demonstrate the usefulness of some approach or as vehicle to explore research issues. The systems are often hand-crafted and manually customised to the needs of individual scenarios. Such solutions can therefore be costly to change. Furthermore, as home care services run as individual applications, conflicts may occur (e.g. two services might wish to control heating levels differently).

Proprietary, off-the-shelf telecare products (for example, http://www.tunstall.co.uk) typically have similar limitations.

Perhaps for this reason, domestic health monitoring and home automation are still not widely deployed. Functionality is usually fixed in special-purpose devices. Data from these devices may not be accessed easily, and the devices may interwork only with others from the same company. The experience of our professional care partners is that adjustments to commercial telecare products are often required.

We have adopted a policy-based (i.e. rule-based) approach for home care to address these issues. Many home care services are reactive applications. Typically, an action is taken when some event happens (e.g. when someone falls or calls for help). A policy-based home care system captures this by specifying the behaviour of a system through policies (i.e. rules). A typical policy in our home care systems consists of an *event*, a *condition* and an *action* (a style known as ECA). When some event happens and the specified condition holds, some action is taken.

By separating service logic from program code, we can modify the behaviour of a home care system without changing its programming. This facilitates configuration and evolution. In our previous work [1], we created a lab prototype and successfully demonstrated a policy-based system using data from a variety of home sensors. Sensor data is used to support a variety of services for home care and home automation. We have since then formally defined the policy language and have implemented this in a policy-based care system. To address compatibility of services, we have also started work on detecting and resolving conflicts among care services [8]. The present paper focuses on the personalisation of care systems.

Section 2 discusses personalisation issues for home care. Section 3 describes the design of a system to support personalisation, section 4 illustrates the approach, and section 5 discusses the implementation. Section 6 reviews related work on this topic. Section 7 reviews the work and describes our future plans.

2. PERSONALISATION ISSUES

A home care system consists of both the technical systems deployed in the home and the external providers of home care. At home, care services react to situations detected by sensors. If necessary, they send appropriate notifications to care centre staff. If required, the system can also monitor the patient's activities of daily living. Lifestyle changes can be inferred from such monitoring, perhaps triggering a review of the care plan. This may lead care staff to change the care services deployed in the home.

Due to differences in individual needs, personalisation needs to reflect a number of factors. Differing health conditions may require different sensors and care services. For example, chronic heart disease might require services for ECG and cardiovascular monitoring, while dementia might require services for reminders and night-wandering.

Care services may also need to reflect personal preferences such as heating, lighting or audio levels. These may have fixed values, or may vary according to the context (e.g. activity, location or time). Thus, the preferred lighting level may depend on whether the user is watching TV or reading.

Other personal preferences may be expressed in terms of constraints over resources available in the environment. For example, the user might wish to refer to a nearby display or to a display with large text. These are examples of context-aware service discovery. The user typically does not have knowledge of the available devices and does not care about the technical details. The user simply needs devices that satisfy certain expectations.

The next section describes a policy-based home care system that supports programming and personalisation of home care delivery through sensors and actuators.

3. POLICY-BASED HOME CARE

In our approach, the behaviour of a care system (i.e. how it reacts to the home and to the user) is specified by policy rules. A home care service is a rule-based application described by policies. We have defined a specialised policy language for this purpose.

Home care policies are defined in *policy documents* which conform to an XML schema. A policy document can contain *policy variables*, *regular policies* and *resolution policies*. Only regular policies and policy variables are discussed here. Resolution policies are a kind of meta-policy that state how to detect and resolve conflicts among regular policies. More details of the policy language can be found in [11].

3.1 Regular Policy

3.1.1 Policy Structure

A *policy* contains a set of *policy rules* that apply to certain subjects. From the functional point of view, there are two major types of regular policies: authorisation policies and obligation policies. An authorisation policy indicates that a subject is authorised to invoke an action upon some target object. An obligation policy specifies that a subject is obliged to take an action upon a target. A policy has a set of attributes that include:

- *applies_to* identifies the entities to which a policy applies. An email-like address is used for entities. In general, a policy applies to a set: one user or entity, one domain, or a list of users, entities and domains. It can also refer to a domain symbolically by citing its variable name. This attribute determines which policy documents are retrieved at run-time.
- *preference* states how strongly the policy definer feels about it, and represents the modality of the policy. This attribute can be used to resolve policy conflicts.
- *enabled* indicates whether a policy can be selected for execution.
- *changed* indicates when the policy was modified.

Policy rules can be combined in various ways, e.g. subject to some condition, tried in sequence, or executed in parallel. Each policy rule consists of three parts: a *trigger*, a *condition* and an *action*.

3.1.2 Trigger

To handle events from sensors, a *device_in* trigger is used with up to five arguments:

- *message_type* for the kind of trigger

- *entity_type* for the kind of entity originating a trigger
- *entity_instance* for the entity instance originating a trigger
- *message_period* for the relevant time period
- *parameter_values* for parameters associated with a trigger.

Some care services may need to react to the situation where time has elapsed since a sensor fired an event (e.g. five minutes after a door was opened). There is therefore a *timer_expiry* trigger for sensor-initiated timers. Such triggers are set up by the policy action *set_timer*. As discussed later, time constraints may also be defined in the condition part of a policy.

3.1.3 Condition

A simple condition consists of a parameter, an operator and a value. Conditions can be combined using logical *and*, *or* and *not* operators. Conditions can refer to variables; this includes environment variables such as the parameters set up by a trigger. The comparison operators are *gt*, *ge*, *eq*, *ne*, *le*, *lt*, *in* (inclusion) and *out* (exclusion). These are interpreted according to the context (e.g. numerical, textual or date comparison).

If a policy does not have a trigger but has a time-based condition, an internal timer is automatically set up by the policy server. Time-based conditions can refer to *day*, *date* and *time*.

3.1.4 Action

To invoke actuators, a *device_out* action is used with up to five arguments. These five arguments have the same names as the parameters of the *device_in* trigger, but have different meanings:

- *message_type* for the kind of action
- *entity_type* for the kind of entity to perform the action
- *entity_instance* for the entity instance to perform the action
- *message_period* for the relevant time period
- *parameter_values* for parameters associated with the action.

There are also internal actions such as:

- *set_variable(name,expression)* and *unset_variable(name)* to change a variable dynamically when a policy is executed
- *start_timer(name,period)* and *stop_timer(name)* for an explicit timer.

3.1.5 Policy Variable

A *policy variable* can be defined in a policy document and used in policy rules. The top-level definition of a variable deliberately resembles that of a regular policy. Each variable has an *id* to identify it uniquely for each owner. The *value* of a variable is what it stands for. As there is only ever one instance of a variable that belongs to the same owner, assigning a new value overwrites the previous one. Variables may hold boolean, numeric or string values. Variables are dynamically typed as in scripting languages, i.e. they may hold different kinds of values at different times. The *owner* of a variable identifies the user or the entity that defined the variable. Normally this will be the same as the *applies_to* attribute, but it could be different if one user (typically an administrator) defines variables that apply to others (typically ordinary users). As an example, an administrator could define the variable *holidays* that listed public holidays for everyone in the current year. Policy variables with the same name but with different owners can co-exist.

As an example, the following policy variable is defined by user Jim for the house where he lives.

```
<variable id="bath_temperature"
 owner="jim@homes.org.uk"
 applies_to="@house5.homes.org.uk"
 value="25" changed="2008-03-23T23:50:10"/>
```

Within certain policy elements, variable names can appear in ranges or in lists. The value of a variable is substituted when a policy is executed, not when it is defined. Variable names can be used in the following cases:

- the *arg* argument of a trigger or an action
- a condition parameter
- an expression
- as the first argument of *set_variable* and *unset_variable*.

3.2 Home Care System Architecture

Figure 1 shows an overview of the home care system.

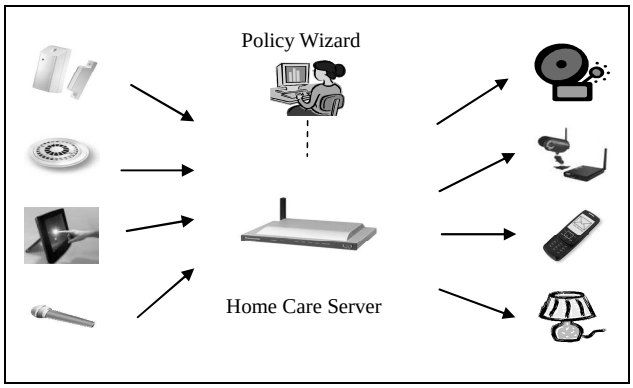

Figure 1. Overview of Home Care System

Sensors installed in the home detect the user's activity or hazardous situations. Commercial products already support these basic requirements, such as fall sensors, movement sensors, and bed occupancy sensors. A widening range of sensors is anticipated in the future. Sensor inputs feed into the home care server, which reacts to the detected situations. Some triggers arise from explicit user input. For example, a pull-cord switch in the bathroom might cause an alarm to be raised in the care centre. Users might also take advantage of accessible interfaces such as touch screens or speech communication. Besides triggers from inside the user's home, remote commands from care staff can also be used to trigger actions.

The actions taken by the home care server include controlling home appliances. Actions can also invoke software applications such as a reminder service, a text message service, or an email service.

The behaviour of the home care system is specified using policies. Inside the system, the policy store holds policies as XML documents. However, the home care system must support non-technical users. The policy wizard is therefore vital as an easy-to-use way of viewing, creating, modifying and deleting policies. One interface supported by the wizard allows policies to be formulated and edited using stylised natural language. This interface is web-based, partly because this is now familiar to many users, and partly because policies can then be edited remotely. However, a textual web interface may not be suitable for all users. Other approaches being evaluated include a speech-based interface and one using digital pen and paper.

A screenshot of the web-based policy wizard is shown in Figure 2.

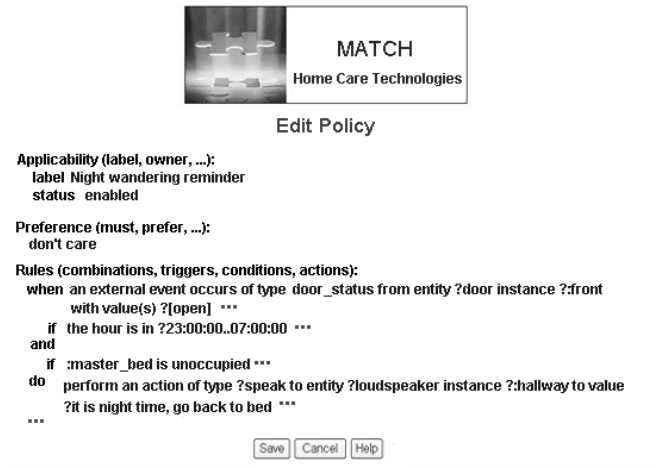

Figure 2. The Web-Based Policy Wizard

The architecture of the home care server is shown in Figure 3.

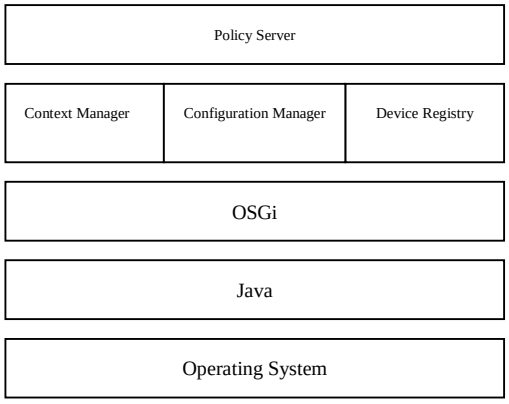

Figure 3. Architecture of The Home Care System

The home care server is built on top of the OSGi platform (Open Services Gateway initiative, *www.osgi.org*). It consists of the following components:

Policy Server On receiving a trigger (e.g. from a sensor), this retrieves relevant policies from the policy store and checks the triggers and conditions of the policy rules. If triggers have occurred and the conditions hold, the corresponding action is put into a list of potential actions. If there is only one action in the list, then this is performed. If there are multiple actions, the policy server checks for and resolves conflicts before initiating the actions. More details about the policy server can be found in [1].

Context Manager When the policy server reacts to a trigger, it also needs to refer to the status of the home care environment (e.g. whether a cooker is on) and the status of the user (e.g. their location). The context manager is responsible for monitoring triggers from sensors and for maintaining the current state of the system.

Device Registry The device registry contains details of currently available sensors and actuators. When sensors and actuators join the system, they register with the system. This can be done manually, but more desirably can be automatically registered through messages to the device registry, as in the Atlas platform [12] or in UPnP (*www.upnp.org*). When sensors and actuators are removed, they are likewise removed from the registry. The device registry provides a query interface to search for devices. Given a set of conditions, the registry can return a list of devices that satisfy the required conditions.

Configuration Manager The configuration manager acts as a broker between the policy server and the device registry. As mentioned in section 2, some care services may refer to resources that are dynamically bound. The configuration manager takes conditions from the policy server and queries the device registry as to devices that satisfy the conditions. This helps to keep the policy rules stable, and saves the care service developer from having to track device status.

4. SUPPORTING PERSONALISATION

A policy-based approach is used for building care services. Apart from hardware deployment, adding or removing a new care service just means adding or removing policy rules. The system allows a different set of policy rules to be created for each user to support their individual care needs. Compared to a program-based approach, the policy wizard is a much more accessible interface for defining how the home care system should react.

For the same care service deployed by different people, the policy language also supports individual customisation. The following section uses policy examples that illustrate how to support personalisation of care services as discussed in section 2.

4.1 Fixed Preferences

Some preferences are represented as constant values. A policy variable definition can specify such preferences. An example of defining preferred bath temperature was given in section 3.1.5.

The following example shows a policy that starts heating bath water at 20:00 every day until it reaches the required temperature. Instead of an external trigger, this example uses a time-based trigger in the condition part of the policy. When this policy is defined, the policy server sets a timer to trigger daily at 8PM. Trigger or action arguments are specified as attributes of the corresponding XML elements. Policy variables are referenced by prefixing their name with ':'.

```
<policy owner=”jim@homes.org.uk”
 applies_to="@house5.homes.org.uk"
 id="Prepare bath water" enabled="true"
 changed="2008-03-15T11:12:03">
  <policy_rule>
    <condition>
      <parameter>time</parameter>
      <operator>eq</operator>
      <value>20:00:00</value>
    </condition>
    <action arg1="on" arg2=":water_heater"
     arg5="temperature=:bath_temperature">
      device_out(arg1,arg2,,,arg5)
    </action>
  </policy_rule>
</policy>
```

The *bath_temperature* policy variable and the policy rule above are specified separately, and may be defined at different times. Since both apply to the same house (*house5.homes.org.uk*), they will be retrieved and used together.

Defining a preference value separately can help in reuse of preference definitions. Policies that refer to a preference variable can share one definition. In addition, different users can have different values for a variable without worrying about interference.

4.2 Conditional Preferences

The home care system also supports conditional preferences through policy variables. These are specified in the conditions of policy rules. In the action part of a policy, different preference values may be defined by using the internal policy action *set_variable*. The variable value can be a literal or a general expression. In both cases, other variables can be referenced.

The following policy example demonstrates how to define conditional preferences using policies. In this example, the user wants to set the TV volume differently depending upon the time of day. The range of the TV volume is from 1 (lowest) to 10 (highest).

The first policy rule shows that the user prefers a louder TV volume during daily activities (9AM to 11PM):

```
<policy_rule>
  <condition>
    <parameter>time</parameter>
    <operator>in</operator>
    <value>09:00:00..23:00:00</value>
  </condition>
  <action arg1="tv_volume" arg2="8">
    set_variable(arg1,arg2)
  </action>
</policy_rule>
```

The second policy rule shows that the user wants to keep a lower volume at night so as not to disturb other people’s sleep.

```
<policy_rule>
  <condition>
    <parameter>time</parameter>
    <operator>in</operator>
    <value>23:00:00..09:00:00</value>
  </condition>
  <action arg1="tv_volume" arg2="3">
    set_variable(arg1,arg2)
  </action>
</policy_rule>
```

The *tv_volume* variable can then be used to adjust the volume of a TV in another policy. At run-time, all related policies are retrieved together and then executed.

4.3 Resource Preferences

Some care services may define constraints on resources available in the environment. For example, the user might want to display a message somewhere in the same room. This is an example of a context-aware service. Without proper support, it would be necessary to discover all the displays in the house before selecting a nearby one. This could require tedious configuration data (or policy rules in our case). Over time, home devices may be replaced and therefore require manual reconfiguration.

In our home care system, policy variables are used to address this issue. Instead of specifying a particular device in policies, constraints on the choice of device are defined in a device variable. At run-time, the policy server passes these constraints to the configuration manager to make a particular choice. The configuration manager finds a suitable device by querying the device registry.

In the following example, the policy variable *nearby_display* is specified for showing messages. The value of this variable contains constraints on the resource: the device function must be a display, and its location must be the same room as the user. The user’s current location is stored in another variable *user_place*. In this example, the location is determined at the granularity of rooms.

```
<variable id="nearby_display"
 owner="jim@homes.org.uk"
 applies_to="@house5.homes.org.uk"
 value="func=display,location=:user_place"
 changed="2007-02-23T23:50:00"/>
```

The following example shows the policy rule for a location-aware message display application. During the day (9AM to 9PM), an SMS message received on the user’s mobile telephone is shown on a display in the same room as the user. The message text is extracted from the *parameter_values* argument of the trigger received from the mobile telephone.

```
<policy_rule>
   <trigger arg1="sms_in" arg2=”mobile”>
     device_in(arg1,arg2)
   </trigger>
   <condition>
     <parameter>time</parameter>
     <operator>in</operator>
     <value>09:00:00..21:00:00</value>
   </condition>
   <action arg1="display"
           arg2=":nearby_display"
           arg5=”content=:parameter_values”>
     device_out(arg1,arg2,,,arg5)
   </action>
 </policy_rule>
```

At run-time, the policy server replaces the *nearby_display* variable with its value before the policy gets executed. Since the value of this variable is a set of conditions, the policy server consults the configuration manager to get the specific device or service that satisfies these conditions.

By introducing a configuration manager, the home care system can now support dynamic resource binding. This hides uninteresting low-level aspects from the care service developer.

5. IMPLEMENTATION

The policy server has been implemented on top of the OSGi platform to support the policy language described in this paper. Knopflerfish (*www.knopflerfish.org*) has been used as the implementation of OSGi 4. Communication between the policy server, sensors and actuators is supported by an event service (*EventAdmin*), provided by the OSGi platform. Wireless sensors from Visonic (*www.visonic.com*) are used to detect conditions such as movement, flooding, smoke, bed occupancy or door opening. A standard wireless receiver has been interfaced to a PC using a USB adapter. An OSGi bundle was written to read the wireless sensor inputs. For output, OSGi bundles have been written to control X10 appliances (on, off and dim actions), to control UPnP alarm devices, to interface with SIP (Session Initiation Protocol, used for Internet telephony), and to send text messages for mobile telephones. More details about the policy system implementation can be found in [1].

The context manager is currently implemented as an OSGi bundle that stores location as policy variables in the policy store on receiving updates from the sensors. The configuration manager is another bundle that currently provides a query-based interface. We plan to add a notification-based interface to inform care services of changes in devices. The device registry is currently implemented using XSet [17], which supports a query interface to an in-memory XML database. The attributes of a sensor or an actuator are specified as name-value pairs in a XML document. Adding devices to the device registry or removing them is currently simulated by adding/removing the XML documents. However, work is under way to integrate the configuration manager with a semantics-enhanced service discovery bundle from one of our colleagues.

The web-based policy wizard is driven by ontologies that contain domain knowledge about home care. The result is a highly flexible user interface, easily adaptable to reflect new applications. For example, the policy system also supports call control in telephony and the management of sensor networks. Existing techniques are being adapted to detect conflicts among home care actions.

6. RELATED WORK

Policy-based management has been applied in a number of areas, including the management of networks and distributed systems [14, 15] and system configuration [16]. However the target users of most policy systems are IT professionals who can be expected to have specialised technical knowledge.

Pervasive (or ubiquitous) computing has attracted considerable research and industrial interest (e.g. [2, 3]). Many approaches to pervasive computing require specialised expertise for customisation or upgrading. Pervasive computing techniques have been applied in clinical settings (e.g. [5, 6]). Although there has been limited research on pervasive computing for care at home, some projects (e.g. [7]) are investigating transmission of medical data back to a care centre.

The Gator project [4, 12] has investigated how to program a pervasive computing space. A key goal is plug-and-play for sensors and nodes. The Gator platform treats sensors and actuators as service objects, and provides an integrated development environment to program these. Plug-and-play could be very useful in our own approach, and may be implemented in future. However, the Gator philosophy is aimed at programmers, unlike our approach which is designed for the less technically minded. In addition, the Gator work does not consider dynamic aspects of the environment at run-time.

The Millennium Homes project [9] investigated multimodal interaction issues to support independent living of older people in their own homes. This work also identified issues with coordinating multiple care services.

[10] proposes a framework to integrate smart home technology with current care practices, focused on temporal reasoning and spatial reasoning.

Preference concepts have traditionally been applied in economics and decision making. Increasingly they have been applied to many other areas. In ubiquitous computing, a personal coordination server has been proposed to filter out nearby devices for mobile users according to their preferences [13].

7. CONCLUSION AND FUTURE WORK

We have described our ongoing work on programming home care systems using a policy-based approach. This supports personalisation and allows easy changes to care services, including adaptation over time. By providing user-friendly tools such as the policy wizard, we have also made the system more accessible to a wider audience – not limited just to developers. We have created the ability to detect and resolve conflicts among care policies, thereby contributing to more predictable and dependable behaviour.

There are a few issues that require further investigation. Easy definition of policy rules may become a concern when more care services are deployed at home. Defining a small set of policy rules in stylised natural language may be acceptable. However as home care systems are more widely deployed, users may find specifying policy rules too tedious. We are looking at techniques to reduce this burden. A promising idea is that of goal refinement, which automatically generates low-level policy rules from a set of high-level intentions.

We also plan to investigate security and privacy aspects of policies in home care systems. Secure definition of policies is already supported in the policy wizard. Management policies will be defined to allow control of services and access to data. It will be crucial to manage what data may be used for what purposes. This includes whether data may be exported out of the home, and how it may be analysed. Health and lifestyle data must obviously be kept confidential, and be processed in an authorised manner by identified individuals.

And at present, policies dictate individual actions or simple combinations of these. For some services it may be desirable to execute complex patterns of actions, in a style similar to workflow modelling. For example, a multimodal reminder service may require a complex set of interactions with the user. One question is whether a complex service should be treated as a whole, or whether the internal flow of a service should be visible. The latter may offer configurability, but can ordinary users handle the complexity? We are investigating the possibilities for integrating workflow-style programming with the policy-based approach.

ACKNOWLEDGMENTS

This research was carried out within the MATCH project funded by Scottish Funding Council under grant HR04016. The authors are grateful for the advice of their collaborators on this project, and also on the PROSEN project (Proactive Condition Monitoring of Sensor Networks).

REFERENCES

[1] F. Wang, L. S. Docherty, K. J. Turner, M. Kolberg and E. H. Magill. Service and policies for care at home, Proc. *1st Int. Conf. on Pervasive Computing Technologies for Healthcare*, pp. 7.1–7.10, IEEE Press, Nov 2006.

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

[3] D. Garlan, D. Siewiorek, A. Smailagic and P. Steenkiste. Project Aura: Toward distraction-free pervasive computing, *Pervasive Computing*, 1(2):22–31, Apr.–Jun. 2002.

[4] A. Helal, W. Mann, H. Elzabadani, J. King, Y. Kaddourah and E. Jansen. Gator Tech Smart House: A programmable pervasive space, *Computer*, 38(3):50–60, March 2005.

[5] J. E. Bardram. Applications of context aware computing in hospital work – examples and design principles, Proc. *ACM SAC '04*, Nicosia, Cyprus, Mar. 2004.

[6] M. Drugge, J. Hallberg, P. Parnes, and K.Synnes. Wearable systems in nursing home care: prototyping experience, *Pervasive Computing*, 5(1):86–91, Jan.–Mar. 2006.

[7] J. E. Bardram. The personal medical unit – a ubiquitous computing infrastructure for personal pervasive healthcare. In T. Adlam, H. Wactlar, and I. Korhonen, eds., Proc. *3rd Ubiquitous Computing for Pervasive Healthcare Applications*, Nottingham, UK, Sep. 2004.

[8] F. Wang and K. J. Turner, Policy conflicts in home care systems, in L. du Bousquet and J.-L. Richier, eds., Proc. *9th Int. Conf. on Feature Interactions in Software and Communication Systems*, IOS Press, Amsterdam, Sep. 2007.

[9] M. Perry, A. Dowdall, L. Lines and K. Hone. Multimodal and ubiquitous computing systems: Supporting contextual interaction for older users in the home. *Trans. on IT in Biomedicine*, 8(3):258–270, 2004.

[10] J. C. Augusto. Towards personalization of services and an integrated service model for smart homes applied to elderly, Proc. *Int. Conf. on Smart Homes and Health Telematics*, pp. 151-158, Sherbrooke, Canada, Jul. 2005.

[11] K. J. Turner *et al*. APPEL: An Adaptable and Programmable Policy Environment and Language, Technical Report CSM-161, Computing Science and Mathematics, University of Stirling, Dec. 2007.

[12] J. King, R. Bose, H. Yang, S. Pickles and A. Helal. Atlas – A service-oriented sensor platform, Proc. *1st Int. Workshop on Practical Issues in Building Sensor Network Applications*, Tampa, Florida, Nov. 2006.

[13] T. Nakajima. Personal coordination server: A system infrastructure for designing pleasurable experience, Proc. *Int. Conf. on Pervasive Services*, 2005.

[14] J. Lobo, R. Bhatia and S. Jaqvi. A policy description language, Proc. *American Association for Artificial Intelligence*, Orlando, Florida, Jul. 1999.

[15] N. Damianou, N. Dulay, E. Lupu and M. Sloman. Ponder: A language specifying security and management policies for distributed systems, Technical Report, Imperial College, London, 2000.

[16] M. Burgess. A site configuration engine, *USENIX Computing Systems*, 8(3):309–337, 1995.

[17] Ben Y. Zhao. The XSet XML search engine and XBench XML query benchmark, Master's thesis, University of California, Berkeley, May 2000.