Feng Wang and Kenneth J. Turner. An Ontology-Based Actuator Discovery and Invocation Framework in Home Care Systems, in M. Mokhtari, I. Khalil, J. Bauchet, D. Zhang and C. Nugent, editors, Proc. 7th International Conference on Smart Homes and Health Telematics, pp. 66-73, LNCS 5597, Springer, Berlin, June 2009

An Ontology-based Actuator Discovery and Invocation Framework in Home Care Systems

Feng Wang, Kenneth J. Turner

Department of Computing Science, University of Stirling, Stirling, FK9 4LA, Scotland
{fw, kjt}@cs.stir.ac.uk

Abstract. Home care systems need to be personalised to meet individual needs, and must be easily adjusted as the user's symptoms develop. Care policies (i.e. Event-Condition-Action rules) can be used to specify care services, facilitating changes in the behaviour of a home care system. Context modelling allows a user to specify the trigger and conditions of a care policy, using high-level context rather than raw sensor data. The actions of a care policy are, however, still dependent on the implementations details of actuators. We propose a framework that allows the actions of a care policy to be specified abstractly using human-understandable concepts. The framework takes care of discovering and using specific actuators, hiding the low-level home networking details from ordinary users. It therefore makes personalisation and modification of home care systems more accessible to ordinary users, requiring very little technical knowledge.

Keywords: Assisted Living, Home Care, Ontology, Service Discovery, Service Invocation.

1 Introduction

Increasingly, providing care at home is seen as a promising alternative to traditional approaches such as nursing homes or sheltered housing. Building and evolving home care systems present significant research challenges. A 'one size fits all' solution is unsuitable. Home care systems are deployed on a large scale, individual care needs will differ, and conditions may change over time. For individuals and for changing circumstances, a home care system must therefore be customisable by non-technical people such as care professionals. Existing home care solutions do not address this issue. In commercial off-the shelf telecare products, functionality is often fixed in special-purpose devices and is not easily customisable. Research prototypes of home care solutions are often hard-coded.

Policy (i.e. rule) based home care systems are promising in making it easier for end users to modify the behaviour of a home care system. A typical care policy consists of an event (i.e. trigger), conditions and actions (otherwise known as ECA). When some event happens, if the conditions hold, then the actions are performed. By changing the rules, the behaviour of the home care system can be modified. Rule-based home care has been investigated in various research projects [1, 2, 3].

In policy-based home care, the events and conditions are often considered together and are defined as context [4]. Home care systems usually do not make use of raw sensor data directly. Instead, raw sensor data is interpreted or aggregated to produce high-level context. The behaviour of a home care system can then be expressed as rules that invoke actuators on context changes. To achieve interoperability among the components of a home care system, an ontology could be used as the basis of a common understanding of context [5]. By using high-level context, care policies need not be affected by differences in specific sensor technologies.

The actions of a care policy, however, are still tied to specific technologies and configurations of the actuators. In a care policy, the actions must state which actuators should perform what operations and with what parameters. The definitions of operations and parameters will vary a lot since different actuators with the same functionality may use different protocols, and services developers may define operations differently. As an example, a lamp could be controlled by EIB/KNX (European Installation Bus) or by X10; these have different APIs. And more complex actuators have an even greater variability in how they are controlled.

Existing solutions to specifying actions often rely on developer-defined service interfaces [6], assume a set of predefined services [1], or consider that an administrator can somehow pass the low-level protocol-specific parameters to the ordinary user who can supply them when editing the rules [2]. None of these is a good solution with regard to deployment of home care systems. For the first approach, actuator developers may define service interfaces differently. For example, two UPnP alarms may be controlled through completely different interfaces. For the second approach, having predefined services limits the deployment of new actuators. For the third approach, the users have to understand some low-level networking protocols. This is because various home network technologies use different operation names and require different sets of parameters for the same functionality. These protocol variations are not hidden from the users. For example, switching on a lamp may use the *On* command in X10, supplying the house code and device code of the lamp as parameters. UPnP may require the method *SetDeviceState* with parameter *DeviceState=on*. Changes in the configuration of actuators (e.g. method of connection or setting) may require changes in care policies. This makes adjustment of home care systems unnecessarily cumbersome and difficult.

We propose an ontology-based framework for actuator discovery and use to enable protocol-independent action specification. Ordinary users have a mental model how an actuator should operate, much as they view driving a car. The user should not have to bother with the technical details of the car, instead just performing standard operations. This concept of common operations should also apply to actuators in home care. The operations of actuators should therefore be modelled in an ontology. Based on this, actuators can register themselves with a semantic service discovery module. Actions in a policy rule can be specified using abstract operations and parameters. At run-time, the semantic discovery module searches for concrete actuator instances. Mapping from abstract actions and parameters to protocol-specific actions and parameters can then take place before executing the actions.

The purpose of our framework is to support end-user programming of home care systems by hiding the technical details of actuators. Just like developers need to write code to infer high-level context, developers also need to write protocol-specific handler code to deal with the mapping of abstract actions and parameters. However, only high-level context and abstract actions are visible to end users.

Section 2 of the paper presents an ontology of actuators for home care. Section 3 describes the design of a system to support actuator discovery and invocation, while section 4 discusses the implementation. Section 5 reviews related work on this topic. Section 6 concludes the paper and describes future work.

2 An Ontology for Actuator Discovery and Invocation

Actuator operations are modelled using the following concepts: *Actuator*, *Service*, *Operation* and *Parameter*, as shown in Fig. 1. An actuator provides some kinds of services. A given service could be also provided by alternative actuator designs. A service supports one or more abstract operations. Each operation has zero, one or more parameters. These concepts are part of a larger home care ontology. For example, *Actuator* is a subclass of *Device* class, which has *Sensor* as a subclass. A device can have properties such as *Location*.

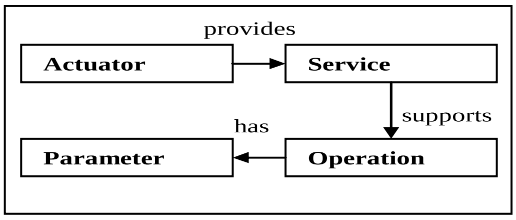

Fig. 1. Core Concepts of The Actuator Ontology

We represent our ontology using OWL (the Web Ontology Language), with each concept represented as an OWL Class. A hierarchy of actuators has also been defined. Subclasses of *Actuator* include *Alarm*, *DVDPlayer*, *Light*, *MobilePhone*, and *TV*. Some subclasses are further divided. For example, *DimmableLight* is a subclass of *Light*. A hierarchy of services is defined similarly. For example, *LightingService* and *DimmableLightingService* are defined, corresponding to services provided by the *Light* and *DimmableLight*.

Since only the abstract operations supported by actuators are relevant, operations are modelled using *Operation* class and parameters using the *Parameter* class. Specific actions supported by the actuators are instances (individuals in OWL terms) of the *Operation* class. For example, *SendSMSTextAction* is an individual of the *Operation* class. *RecipientNumber* and *TextContent* are both individuals of the *Parameter* class that could be used by *SendSMSTextAction*.

An OWL class can also have properties, typically a *DataType* property or an *Object* property. The first links an individual with data values while the second links one individual with another individual. For example, the Object property *hasLocation* links a *Device* individual with a specific *Location* individual.

To encode low-level protocol details about how to invoke a specific actuator, a *DataType* property is defined for an *Actuator* class. The *hasRAI* property links an

Actuator individual with a string that represent the resource access identifier (RAI) to access the service provided by an actuator. This property defines the protocol and device (or service) identifier, separated by colon. This identifier uniquely distinguishes actuators using the same protocol and will correspond to different values in various home networks. For example, the RAI of an X10 living room light might be 'X10:b1', identifying the house code and device code. The RAI of a UPnP alarm might be "UPnP:fallAlarm", identifying its UDN (Unique Device Name).

3 Architecture of Actuator Discovery and Invocation

The overall architecture of policy-based home care is depicted in Fig. 2. Users specify care policies using a *Policy Wizard*. Sensor wrappers gather raw sensor data from sensors, convert them into entity properties (similar to the sensor wrappers in [7]), and store them into a knowledge base. The context server performs further reasoning based on facts in the knowledge base, and generates high-level context information. Changes in entity properties (i.e. context changes) are sent through an event service to the policy server. The policy server checks incoming events and environment state against the triggers and conditions of the rules; this decides whether to execute the policy actions. If execution is required, the policy server sends commands through event services to invoke actuators. (The policy server also performs other tasks as such refining high-level goals into policies and handling conflicts among policies.)

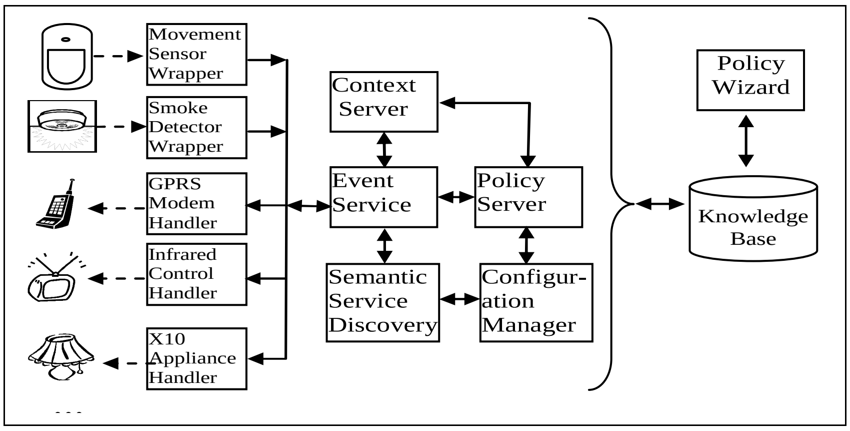

Fig. 2. System Architecture for Policy-Based Home Care

To discover specific actuators and protocol-specific parameters, the policy server consults the configuration manager and the semantic service discovery module. The event-based communication paradigm decouples sensors, actuators and the policy servers, so that changes in one of them will not affect others. The following section describes actuator discovery and invocation in detail. Using an ontology in service discovery and invocation involves the following steps: registering actuators, specifying actions in a policy rule, and run-time invocation of actuators.

3.1 Registering Actuators

When an actuator is installed in a home care system, its capabilities are registered with the semantic service discovery module and stored in the knowledge base. The latter represents actuators as individuals of specific classes. For example, the description of a lamp in the living room includes the actuator class it belong to (*DimmableLight*), its location (*LivingRoom*), its RAI (X10:b1), its manufacturer and other information.

Actuators may use different types of home networks. Some network technologies have built-in discovery mechanism. For example, UPnP defines how devices can be discovered and controlled programmatically using an API. Software agents can listen for UPnP discovery events and automatically register/unregister actuator capabilities in the semantic service discovery module. For actuators using home networks without a service discovery mechanism, a graphical tool can allow users to manually register devices in the knowledge base.

3.2 Specifying Actions

The policy wizard makes it easy to specify care policies. To specify an action, the following elements need to be specified: actuator, action and parameters. Specifying an actuator using a concrete RAI value essentially hard-codes which specific one is used. Instead, a set of conditions is specified that an actuator must satisfy. Currently the conditions are stored in a device variable held by the policy system. A device variable has a name and a definition that acts as a placeholder for a specific value. The name of the variable is referred to in a policy action. The conditions include the class that the specific actuator belongs to and a set of property restrictions.

A SPARQL query string is generated to represent the conditions specified by the user. (SPARQL, http://www.w3.org/TR/rdf-sparql-query, is the language used to query RDF-based documents.) Below is a sample SPARQL query to find the RAI of a living room light that can be dimmed:

```
SELECT ?RAI
WHERE {
 ?ins a configuration:DimmableLight.
 ?ins configuration:hasLocation
    configuration:LivingRoom.
 ?ins configuration:hasRAI ?RAI.
 }
```

Based on the class of the desired actuator, the policy wizard retrieves from the ontology the supported abstract operations and associated parameters. These are presented for the user to make a choice. This makes sure that only actions and parameters supported by an actuator are specified in the policies. This helps to avoid errors and reduces the user’s learning burden.

3.3 Run-time Execution of Actions

Three steps are followed when a policy action is performed: find the specific actuator to execute the action, map the action and parameters to protocol-specific ones, and finally execute the action.

The semantic service discovery module executes the SPARQL query defined by the device variable. The RAI property of the actuator found is extracted. The policy server then constructs a system event using the RAI, the abstract operation and the parameters. The protocol and the actuator identifier of the RAI are stored in the event properties. Finally the policy sends out this event through the event service.

Each home network protocol handler registers its interest in certain type of events by supplying an event filter to the event service. The event filter includes a condition on the protocol property of an event, so an individual handler receives only events associated with its protocol (e.g. *protocol=UPnP*). The *device_identifier* property of an actuator event is used by the handler to find the specific actuator to execute the action.

Individual actuator handlers also map abstract actions and parameters to protocol-specific ones before execution. Although these handlers have the same interface structure, the complexity of mapping the action and parameters varies a lot according to the network technology. For home networks such as X10, the semantics of operations are straightforward, so the mapping is easy. For other home networks, an automatic mapping may be difficult. For example, various developers may define differently the operations to activate an UPnP alarm. They may give different names to the action and may also use different state variables as parameters. Furthermore, one abstract operation may map to several UPnP actions. It is therefore necessary to rely on a protocol-specific actuator event handler to perform these mappings. Developers need to write specific code for this. The users of policy wizard do not need to be aware of this mapping. In order to specify the rules, they only need to know the abstract concepts present in the common ontology. Architecturally, when an actuator is installed into a home care system, the corresponding actuator event handler needs to register with the event service and be ready for the execution of protocol-specific actions.

4 Implementation and Evaluation

The policy-based home care system has been design using the OSGi platform as the basis (http://www.osgi.org). Knopflerfish (http://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 OSGi. Wireless sensors from Visonic (http://www.visonic.com) are used to detect conditions such as movement, flooding, smoke, bed occupancy and door opening. A standard wireless receiver has been interfaced to a PC using a USB adapter.

The ontology has been developed using Protégé (http://protege.stanford.edu). Semantic service discovery and invocation has been implemented using the Jena semantic web framework (http://jena.sourceforge.net) and integrated with the existing policy server. The following protocol-specific event handlers have been created for mapping abstract to protocol-specific levels, and for executing actions:

- *X10* for controlling home appliances using an X10 network
- *UPnP* for controlling devices in a UPnP network
- *IRTrans* for controlling audio and video equipment such as TVs and DVD players using the IRTrans® infrared control system
- *SMS* for sending and receiving text messages using a GPRS modem.

The policy wizard has been enhanced to support abstract actions using concepts in the ontology. By using SPARQL queries to search against the knowledge base, the policy wizard can make use of the following information:

- the actuator classes available
- the properties of each actuator class
- the individuals in an actuator class that satisfy certain conditions over properties
- the operations supported by an actuator
- the parameters used by a given action.

To specify the actuator required, the user first selects a specific actuator class and then specifies conditions on properties of that class. For actions, the user chooses from the list supported by that actuator. Similarly, parameters are selected from those applicable to an action.

The built-in OSGi service discovery mechanism uses exact type matching. The semantic discovery module described here can find actuators based on subtype-supertype relations. Thus the search result for individuals of the *Light* class will also include those of the *DimmableLight* class.

By making use of the ontology, policy creation becomes easier because users can select alternatives rather than input them manually. Policy editing also uses familiar concepts such as ‘living room light’ rather than unfamiliar ones such as an X10 or UPnP address. Since the actuator can readily be extended with new definitions, the ontology-based specification of policy actions makes the approach more extensible.

5 Related Work

There have some efforts towards rule-based home care systems. [1] proposes a framework to integrate smart home technology with current care practices, focusing on temporal reasoning and spatial reasoning. [10] discusses a three-tier general architecture to support end-user programming of home care systems. Both work do not address actuator integration and discovery. The Gator [6] platform treats sensors and actuators as service objects, providing development environment to programmers, unlike our approach which is designed for the less technically minded. In addition, the Gator work does not consider dynamic aspects of the home care environment.

Semantic web technologies have been applied in pervasive computing environments to achieve interoperability among heterogeneous systems. In particular, ontologies have been used to model context and reason in pervasive computing environments [7, 5, 8]. [11] presents a user interface level context model for assistive living. However, its focus is on context modelling, not on actuator part.

6 Conclusion and Future Work

A framework has been presented for actuator discovery and invocation in home care systems. By making use of an ontology to model the services and operations of actuators, policy actions are made protocol-independent and are not affected by changes in home network configuration due to the evolution of home care systems.

Future work will extend the implementation to look at the following issues:

- Tools will be created to register devices as individuals in the knowledge base. Currently, adding/removing devices requires manipulation of OWL documents. For actuators with no associated service discovery mechanism, a simple solution would be a graphical tool for registering actuators manually. However, a promising approach could be using the smart-plug concept [9] to automatically register the functionality of actuators.
- The existing context ontology will be integrated with the actuator ontology, so that the user can specify both the situations and corresponding actions abstractly. The framework would automatically take care of mapping and execution. This would hide from users the uninteresting low-level configuration details of sensors and actuators. Evaluation of usability and acceptance of this approach will follow.

Acknowledgments. The authors gratefully acknowledge the support of the Scottish Funding Council SRDG MATCH project HR04016.

References

1. J. C. Augusto. Towards personalization of services and an integrated service model for smart homes applied to elderly, in *Proc. Int. Conf. on Smart Homes and Health Telematics*, pp. 151–158, Sherbrooke, Québec, Canada, Jul. 2005 (ICOST 2005).
2. F. Wang *et al*. Towards Personalised Home Care Systems, in *Proc. Int. Conf. on Pervasive Technologies related to Assistive Environments*, pp. L2.1-L2.7, ACM, July 2008.
3. K. Du *et al*. HYCARE: A hybrid context-aware reminding framework for elders with mild dementia, ICOST 2008.
4. A. K. Dey, D. Salber and G. D. Abowd. A context-based infrastructure for smart environments. In *Proc. 1st Int. Workshop on Managing Interactions in Smart Environments*, pp. 114–128, Dublin, Dec. 1999.
5. X Wang *et al*. Semantic Space: An infrastructure for smart spaces, *Pervasive Computing*, 3(3): 32–39, 2004.
6. 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.
7. A. Ranganathan et al., A middleware for context-aware agents in ubiquitous computing environments, in *Proc. Int. Middleware Conference*, Rio de Janeiro, Brazil, Jun. 2003.
8. H. Chen *et al*. An ontology for a context aware pervasive computing environment, in *Proc. Workshop on Ontologies and Distributed Systems*, Acapulco, MX, Aug. 2003.
9. H. Elzabadani et al., Self-sensing spaces: Smart plugs for smart environments, ICOST 2005.
10. T. Zhang *et al.* Empowering the user to build smart home applications, ICOST 2004.
11. M. Wojciechowski *et al.* A user interface level context model for ambient assisted living, ICOST2008.