Realizing Networks of Proactive Smart Products

Mathieu d'Aquin, Enrico Motta, Andriy Nikolov, and Keerthi Thomas

Knowledge Media Institute, The Open University, MK7 6AA, Milton Keynes, UK
{m.daquin, e.motta, a.nikolov, k.thomas}@open.ac.uk

Abstract. The sheer complexity and number of functionalities embedded in many everyday devices already exceed the ability of most users to learn how to use them effectively. An approach to tackle this problem is to introduce 'smart' capabilities in technical products, to enable them to *proactively* assist and co-operate with humans and other products. In this paper we provide an overview of our approach to realizing networks of proactive and co-operating smart products, starting from the requirements imposed by real-world scenarios. In particular, we present an ontology-based approach to modeling proactive problem solving, which builds on and extends earlier work in the knowledge acquisition community on problem solving methods. We then move on to the technical design aspects of our work and illustrate the solutions, to do with semantic data management and co-operative problem solving, which are needed to realize our functional architecture for proactive problem solving in concrete networks of physical and resource-constrained devices. Finally, we evaluate our solution by showing that it satisfies the quality attributes and architectural design patterns, which are desirable in collaborative multi-agents systems.

Keywords: Proactive Problem Solving, Smart Products, Knowledge Systems, Ontology Engineering, Distributed Problem Solving.

1 Introduction

The sheer complexity and number of functionalities embedded in many everyday devices already exceed the ability of most users to learn how to use them effectively. This is not just the case for mass consumer devices, such as mobile phones, but it also applies to work settings, where the increased diversity within product lines introduces new complexities in both product assembly and maintenance processes [1]. An approach to tackle this problem is to introduce 'smart' capabilities in technical products, to enable them to better assist and co-operate with humans and other products [1]. In the context of the *SmartProducts* project[1] we have investigated these issues in a number of scenarios, drawn from the aerospace, car and home appliances industries. In particular, a key requirement for *smart products* imposed by our scenarios is that they need to be able to exhibit *proactivity*, both to improve the level of assistance to users and also to be able to co-operate effectively with other smart products in shared task scenarios. Proactivity can be informally characterized as the ability of a smart product to take initiative and perform some action, without having been specifically instructed to do so by a user or another smart product [2]. In addition, in order to be able to form and join networks with other smart products and engage in co-operative problem solving, smart products must also be capable of *self-organisation*. This second requirement is a cornerstone of our approach, to ensure that our solutions can be applied in

[1] http://www.smartproducts-project.eu/

open, networked scenarios, where a closed, top-down design of co-operative problem solving solutions would be too restrictive, if not unfeasible [1].

While the SmartProducts project has looked at a whole range of issues, which need to be tackled to support effective networks of context-aware and proactive smart products, including *user interaction* [3], *access control models* [4], and *distributed storage* [5], in this paper we focus on the core challenge posed by the project scenarios: *the design and implementation of a computational infrastructure to realize networks of proactive smart products*. Hence we will provide a complete overview of our approach, covering both the *knowledge level* and the *symbol level* [6] elements of our solution and in particular showing how we have integrated semantic technologies with ubiquitous computing solutions, to equip resource-constrained physical devices with the capability of representing and reasoning with knowledge structures and engaging in co-operative problem solving.

Specifically, the discussion in the paper will cover: i) our characterization of the notion of 'proactive behaviour', ii) the ontological basis of our framework, iii) a knowledge-level, task-centric problem solving architecture for characterizing co-operative and proactive problem solving in networks of smart products, and iv) a concrete realization of this functional architecture on networks of Android devices. In addition, we will also show that v) our approach to proactive, distributed problem solving satisfies the quality attributes and architectural design patterns, which are desirable in collaborative multi-agents systems [7].

We will start the discussion in the next section, by introducing one of three real-world scenarios which have motivated the work in the SmartProducts project, which will then be used in the rest of the paper to illustrate our solutions.

2 A Scenario

The scenario we consider in this paper is one in which a user, whom we refer to as Ashley, is organizing a dinner for her friends. To this purpose she uses a *Cooking Assistant* application, available on a tablet device, which allows her to specify the parameters for the event, including dietary requirements, food preferences, the number of people attending, the date of the event, known people attending, etc. Ashley's *Smart Kitchen* also includes a *Smart Fridge* and *Smart Cupboards*, which are aware of their contents and can contribute to the shared task scenario –e.g., by providing information about items which are due to expire soon, thus minimizing food wastage. Another smart product available to Ashley is the *Shopping Assistant*, an application[2] which maintains a history of food purchases and is also able to provide information about currently discounted items at a supermarket of choice, which may be relevant to the current meal plan.

Once Ashley has selected a *meal plan* from the various suggestions provided by the *Cooking Assistant* with the co-operation of other smart products, a *shopping list* is produced by the *Shopping Assistant*, on the basis of what is required by the meal plan and what is already available in the house.

[2] In this paper we use the term 'smart product' to refer both to physical products, such as a smart fridge, and also to software products, such as a meal planner, which is an application running on a particular device. Here, it is important to emphasize that we do not use the term 'smart product' to refer to generic computing platforms, such as a tablet device. These are characterized instead as 'containers', in which numerous smart (software) products may be installed.

Once at the supermarket, the *Shopping Assistant* interacts with *Supermarket Agents* to identify the best deals for the items in the shopping list. However, while Ashley is shopping at the supermarket, a member of the family removes the bowl of strawberries (an ingredient needed for one of the selected recipes) from the fridge. At this point Ashley receives a notification from the *Smart Fridge* that the strawberries are no longer available, and therefore she can choose to add strawberries to the shopping list.

Hence, this scenario requires cooperation between a number of smart products, specifically: *Cooking Assistant, Smart Fridge, Smart Cupboards, Shopping Assistant*, and *Supermarket Agents*, with the interactions between these smart products occurring around shared tasks in two different *ambiances*[3], a *Smart Kitchen* and a *Supermarket*. The requirement for proactivity here means that the smart products must be able to proactively contribute to shared tasks, such as meal planning and shopping at the supermarket, both at the time when requests for information are broadcast but also, as in the example of the strawberries being removed from the fridge, when some event occurs which has implications for the tasks currently being executed.

3 Proactivity in Networks of Smart Products

For an agent to be *proactive*, it needs to be able to take action in scenarios in which such action has not been explicitly programmed in the agent, or explicitly delegated to the agent by a user or another agent. An approach to realizing proactive behaviour entails the development of *personal assistants* [8], which are able to monitor and learn from user behaviour, to anticipate their needs and exhibit proactive assistance. However, in the context of our project, we are less interested in these *application-specific* [2] solutions, than in enabling proactivity in open scenarios, where proactive problem solving takes place in the context of networks of smart products.

So, what are the requirements for proactivity in such scenarios? As discussed in [9], in co-operative problem solving scenarios *"the ability to anticipate information needs of teammates and assist them proactively is highly desirable…"*. In particular, the ability to proactively provide task-related information to a task-performing smart product is especially useful in an open network of smart products, because it avoids the need to know a priori who can provide certain information or carry out a particular task, thus allowing the dynamic formation of networks of co-operating smart products. Hence, a key type of proactive behaviour we want to support concerns precisely this *ability of a smart product to proactively contribute information to another smart product in a task-centric context*.

In addition, achieving proactive, co-operative problem solving and self-organization requires flexible mechanisms for representing and manipulating problem solving knowledge. Since networks of smart products do not have a predefined organizational structure with prescribed roles, and the capabilities of involved products may vary, smart products can contribute to shared tasks in various ways. For example, the meal planning task presented in Section 2 will be executed differently depending on which smart products are part of the *Smart Kitchen ambiance*, and their capabilities. Hence it is undesirable to design a complete process decomposition in advance and we require instead mechanisms for dynamic process composition, task assignment and proactive problem solving.

---

[3] An ambiance denotes an environment comprising specific smart products, in which collaborative problem solving can take place – see Section 3 for more details.

An *ambiance* denotes an environment comprising specific smart products, in which collaborative problem solving can take place. While in many cases ambiances reflect physical spaces, such as the ambiance comprising all smart products in a kitchen, the notion itself is flexible and simply denotes any collection of smart products which come together to engage in co-operative problem solving. Both the inclusion of products into an ambiance and the set of permissible behaviours are regulated by *joining policies*. For instance, within a supermarket ambiance, it may be desirable to allow smart products belonging to customers (i.e., on mobile devices) to join the ambiance and receive information, adverts and recommendations from supermarket agents (or even directly from smart food items), but these devices may not be allowed to expose arbitrary tasks to other customers' devices.

In sum, in order to comply with the requirements from our scenarios, the key aspect of proactive behaviour we wish to support concerns the ability of smart products to engage in distributed, collaborative, proactive problem solving, including the ability i) to expose and be aware of shared tasks, ii) to proactively provide task-specific information to other smart products in a particular problem solving context, and iii) to provide actual solutions through problem solving for those tasks which fall within a smart product's set of capabilities.

4 Semantic Technologies for Smart Products

4.1 The SmartProducts Network of Ontologies (SPO)

The *SmartProducts Network of Ontologies* (*SPO*)[4] has been developed both to provide a clear specification of the conceptual model underlying the work on the SmartProducts project, and also to maximise interoperability not just among all SmartProducts applications, but also between these and other applications in related domains. SPO comprises three different sets of modules, which reflect different levels of abstraction and reuse, from the most generic to the application-specific ones. Because each layer is itself divided into several sub-modules, we obtain a highly modular design, which makes it possible to reduce the parts to be used by a device, depending on its functionalities –e.g., no need to use the process model on devices that only serve as data providers.

Figure 1 shows the configuration of SPO, which was used for the smart kitchen scenario. The other project scenarios were modelled in a similar way, reusing the external and generic modules and plugging-in alternative application-specific ones.

SPO builds on established external ontologies, including the DOLCE Ultra Lite Ontology (DUL)[5], which provides the foundational layer to interpret SPO entities. For example, SPO reuses the *Quality-Region-Parameter* design pattern from DUL to model physical qualities of objects, and the *Plan-PlanExecution* pattern to distinguish between the definition of problem-solving behaviour and its actual execution. Concepts and properties dealing with representation of time are imported from the W3C time ontology[6].

An essential part of SPO covers the *context model*, which is needed to ensure that smart products are able to assess the available information about the state of the world. In particular, the context model includes both 'low-level' contexts, i.e., atomic facts about the state of the environment and its changes –e.g., smoke being detected, as well as 'high-level'

[4] http://projects.kmi.open.ac.uk/smartproducts/ontologies/.
[5] http://www.loa-cnr.it/ontologies/DUL.owl
[6] http://www.w3.org/2006/time

ones, i.e., abstracted information about a situation as a whole, which can be inferred from aggregated low-level context facts –e.g., an emergency situation being recognised on the basis of smoke detection. Sensing devices, which can be either embedded –e.g., a thermometer in a fridge, or external –e.g., a weather service, serve as sources of low-level context information. To model their output, SPO utilizes the recently proposed *Semantic Sensor Network (SSN)*[7] ontology, in particular by reusing its *Observation* design pattern. This allows external sources of sensor data relevant to the products within an ambiance – e.g., readings provided by a weather service, to be smoothly integrated and used alongside data produced by embedded sensors.

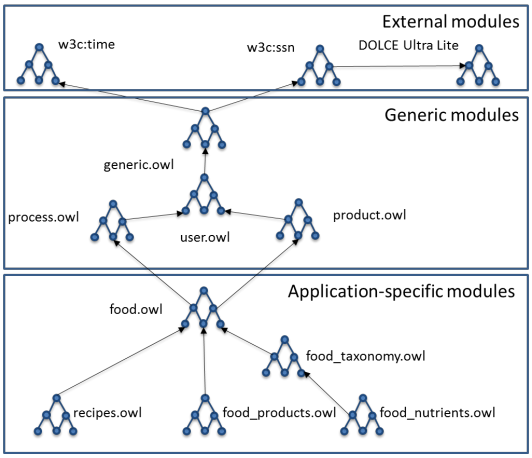

Figure 1. The SmartProducts Network of Ontologies configured for the smart kitchen scenario

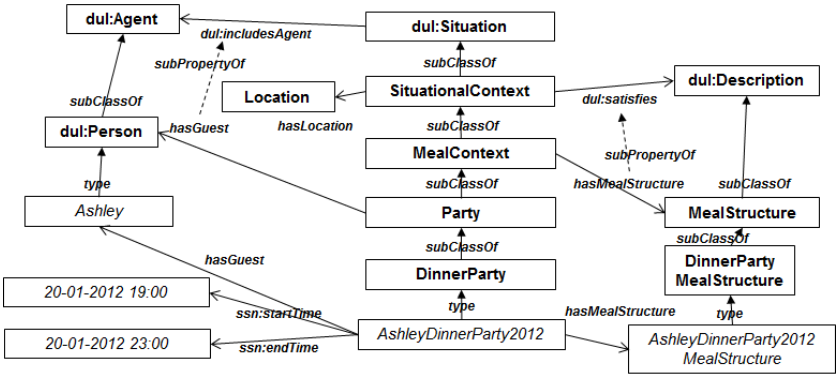

Figure 2. Modelling situational context

To model a high-level situational context, which is usually abstracted from low-level context data, we reuse the design patterns developed in situational awareness research [10] and we characterize a situation as an abstract interpretation of a specific state of the environment. Examples of situations include a dinner party, a snowstorm on the driving route, or a visit to a car workshop. Situations are characterized by their space and time

[7] http://purl.oclc.org/NET/ssnx/ssn

constraints, participating objects, specific relations between these objects, and the situation type [10]. Depending on the situation type, specific types of roles for objects and relations can be defined. For example, a meal can have one or more guests, while a car service operation may involve the car owner, one or more technicians, and a manager. In SPO, the class *SituationalContext* (see Figure 2) defines generic descriptions for high-level contexts. It extends the class *dul:Situation*, by localizing its instances in space and time. Specific properties determining different types of relations between entities and situations are defined by reusing relevant subproperties of the generic *dul:isSettingFor* relation –e.g., *dul:includesAgent*, *dul:includesEvent*, etc.

4.2 Task-based Problem Solving Architectures

Research on generic models of problem solving in knowledge engineering has developed a number of libraries, architectures, languages and tools to support the specification of generic and reusable problem-solving components [6 11 12 13 14]. A key feature of these architectures is that they support a strong separation of the different building blocks for intelligent systems, for example, distinguishing between *task*, *method*, *domain* and *application knowledge* [11]. Thus, they provide a sound epistemological and architectural basis for developing robust knowledge systems by reuse. In particular, here we build on the *problem solving architecture*[8] defined by the *TMDA framework* [11], which provides a rich modelling framework, based on *task* and *method ontologies* [11], which supports the specification of problem solving components in detail.

Although the TMDA architecture was originally conceived for 'closed' application scenarios in knowledge-based systems, it has been modified in recent years to provide the basis for an open distributed semantic web service architecture [15]. However, our scenarios impose new requirements on the TMDA architecture, as methods and tasks are executed and exposed by specific smart products, and problem solving takes place in specific ambiances, where specific policies are enforced. Hence, as a first step we adapted the TMDA framework to the SmartProducts scenarios, as discussed in the next section.

4.3 Ontological Modelling of Problem Solving Knowledge

A key element of SPO is the ontological support for modelling proactive and co-operative problem solving in networks of smart products and to this purpose SPO extends the *task ontology* provided by the TMDA library [11], by introducing the concepts needed to characterize smart products and ambiances and integrating these notions with the modelling of tasks and problem solving methods. At the core of the SPO ontology is the concept of *TaskInAmbiance* (see Figure 3), which is defined in terms of *input* and *output roles*, a *goal specification*, the *ambiances* in which it is exposed, and an optional *closing time* and *closing condition*. Input and output roles define the information types specifying the input and output of a task. A goal defines a condition that the output needs to fulfil, in order for the task to be achieved, and it is represented in the ontology as a subclass of *owl:ObjectProperty*, i.e., as a meta-property. This representation allows us to use specific instances of class *GoalProperty*, i.e., specific goal-defining object properties, to represent the goal of a particular task. The optional closing time and closing condition are used to

[8] A problem solving architecture focuses on knowledge-level components for problem solving, in contrast with a system architecture, which concerns technical design issues – see Section 5.

specify precise end points (as a time point or as a logical condition) for other smart products to consider, if they are willing to provide information or tackle the task. In particular closing conditions are also modelled as meta-properties, using the same mechanism used for representing goals. If no closing time or condition are given, the goal specification provides a default closing condition for a task.

A task is solved by a *Problem Solving Method (PSMInAmbiance)*, which defines a procedure to solve a class of tasks and is defined as a subclass of *dul:Plan*. An *ApplicabilityCondition* can be specified for a PSM[9] to determine whether it can be applied to a task in a specific problem solving context. An *ApplicabilityCondition* for a PSM is defined as a relation object with two arguments: *TaskType*, which defines the class to which the method can be applied (either the actual class or a set of restrictions describing a class definition) and *AmbianceType*, which defines a class of ambiances in which the method can be executed. *TaskType* and *AmbianceType* are defined as meta-classes: i.e., their instances are themselves classes, which are subclasses of *Task* and *Ambiance* respectively. For instance, an applicability condition may refer to the class of tasks *MealPlanningTask* and restrict the class of ambiances to those which belong to the specific user (i.e., have the value of *hasOwner* property set to a specific user ID).

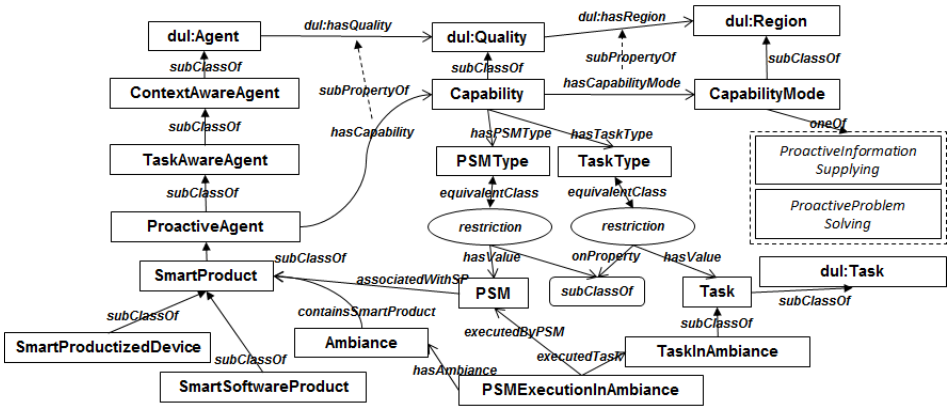

Figure 3. Main classes and relations for modelling problem solving knowledge

A *SmartProduct* is defined as both a *Product* and a *ProactiveAgent*. Two subclasses of *SmartProduct* are considered in the ontology, *SmartProductizedDevice* and *SmartSoftwareProduct*, to cater for both physical and software products, as pointed out in Footnote 2. A relation, *hasCapability*, is used to define a *Capability* for a *ProactiveAgent* (and therefore for a *SmartProduct*). A *Capability* is defined as a quality of an agent and represented as a tripartite relation object <*PSMType, CapabilityMode, TaskType*>, where *TaskType* describes the class of tasks the agent can contribute to solve, *PSMType* describes the method the agent in question will apply, to tackle the instance of *TaskType*, and *CapabilityMode* specifies the modality by which the agent can contribute to solving the task. Currently, we consider three types of *CapabilityMode:*

- *ProactiveProblemSolving*. This value specifies that the method can be applied directly to solve the task.

[9] For the sake of brevity, in the rest of this paper we will use the terms *Task* and *PSM* as synonyms for *TaskInAmbiance* and *PSMInAmbiance*.

- *ProactiveInformationSupplying*. This value specifies that the method provides information relevant to the task execution, by modifying some aspect of the task – typically the input roles.
- *TaskExposing*. This value refers to the generic capability of an agent to expose a task to an ambiance and is associated with a default method for exposing tasks.

The class *Ambiance* is used to specify networks of smart products, given that collaboration between smart products is only allowed within a particular ambiance. The following properties are defined for class *Ambiance*:

- *containsSmartProduct:* links the ambiance to a smart product, which is currently in the ambiance in question.
- *hasOwner:* links the ambiance to its (human) administrator.
- *hasJoiningPolicy:* links the ambiance to a joining policy descriptor. There can be several descriptors defined for the same ambiance.

Joining policies are necessary in order to regulate the inclusion of products into the ambiances. For example, the owner might not want products belonging to non-trusted users to join her home ambiance. Moreover, she may want to restrict certain capabilities of products within it. For instance, within a supermarket ambiance, it may not be desirable to allow smart products belonging to customers (mobile devices) to advertise arbitrary tasks to other customers' smart products.

5 Realizing Networks of Proactive Smart Products

Here we describe how the conceptual framework presented in Section 4 (the *knowledge level*) has been realized at *symbol level*. In particular, we focus on two critical technical design issues: i) the realization of a protocol implementing proactive, distributed problem-solving over networks of peer-to-peer devices, and ii) the realization of the semantic data management components needed to store and manipulate knowledge in smart products.

5.1 The SmartProducts Task Messaging Protocol

To achieve a peer-to-peer model where all smart products are connected to each other in a network, we implemented a system architecture where smart products are instantiated as applications deployed on mobile computing devices, specifically smartphones and tablet devices running Android OS. For the actual communication between smart products, we chose the *MundoCore* [16] communication middleware, which provides facilities, such as creation of network *zones,* which can be readily mapped to the concept of ambiance in our framework. MundoCore also supports the notion of *channels* and implements a *publish/subscribe mechanism* to enable agents to interact and exchange messages within a zone. These features (zones and channels) were used to realize ambiances as networks within which tasks, their outputs and related information can be exchanged between smart products. Moreover, MundoCore also allows devices to subscribe to multiple channels in multiple zones, therefore allowing smart products to participate in more than one ambiance at a time –e.g., the *Shopping Assistant* in our scenario can be at the same time in the kitchen and in the supermarket ambiance. Finally, MundoCore supports 'device discovery' in such a way that it allows devices to join and leave the network at any time without having to reconfigure the network or the devices.

Besides the straightforward reuse of the zone and channel mechanisms, the realization of the distributed, proactive problem solving approach described in the previous sections was achieved by implementing a dedicated protocol on top of MundoCore, which we refer to as the *SmartProducts Task Messaging Protocol* (*SPTM*). SPTM implements the proactive task-based problem solving approach described earlier by relying on a coordination mechanism similar to the one used in *contract-nets* [17].

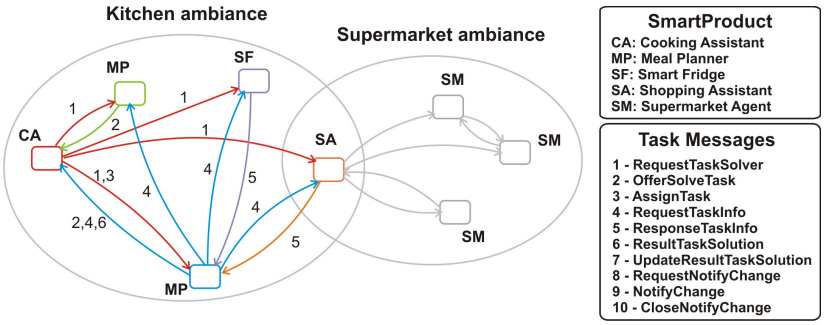

Figure 4. SmartProducts Task Messaging protocol

As shown in Figure 4, SPTM provides mechanisms i) to broadcast and assign tasks; ii) to request and gather contributions to tasks; iii) to handle the management of tasks within multiple ambiances –e.g., the *Shopping Assistant* is able to receive messages in both the kitchen and supermarket ambiances; and iv) to react to events affecting ongoing tasks in an ambiance –e.g., a change in the content of the fridge triggering an alert. In particular, Figure 4 shows a subset of the flow of messages in our scenario, where the *Cooking Assistant* first broadcasts a meal planning task (type *RequestTaskSolver* in the figure), to which the *Meal Planner* responds by sending a message of type *OfferSolveTask*. The *Meal Planner* is then delegated to tackle the meal planning task (type *AssignTask* in the figure) and to this purpose it broadcasts a message of type *RequestTaskInfo*, to which the *Smart Fridge* responds by providing information about its contents (type *ResponseTaskInfo*).

Since tasks are sent to the environment without pre-compiled knowledge or assumptions about the capabilities of other smart products, they carry associated closure conditions –in the simplest form, a time delay. The protocol is relatively lightweight, as it relies on only 10 types of messages (6 of which are shown in the diagram on the left-hand-side of Figure 4), and we have not observed any significant messaging overheads compared to similar designs based on the contract-net protocol.

The implementation of the conceptual framework for distributed proactive problem solving into the concrete SPTM protocol allows us to achieve a number of useful properties, as will be discussed in detail in Section 6. In a nutshell, this implementation reflects the flexibility of the framework, making it possible to add smart products to an ambiance without any need for reconfiguration of the network. The implementation also optimizes the distribution of knowledge amongst smart products, as information is stored locally in each node of the network and exchanged only when needed for a specific task.

5.2 Semantic Data Management Infrastructure

A key issue related to the realization of our approach in concrete networks of smart products concerns the need to equip resource-limited devices, such as mobile phones, with

the ability to store and reason with semantic data. In [18] we compared different frameworks for semantic data management on resource-limited devices, and showed that small to medium scale data could be handled adequately on modern smartphones. In particular, we chose the *Sesame*[10] triple-store for its low requirements in terms of memory usage. Unsurprisingly, this study also showed that computational demands increased linearly with the amount of data stored, and with the use of advanced features, such as embedded inference engines. Hence, this study provided us with a basis to assess the amount of resources needed for a particular smart product depending on its required local knowledge and capabilities. For example the *Smart Fridge* only requires the limited resources that can be found on a smartphone, while the *Cooking Assistant*, which applies potentially complex inferences on thousands of cooking recipes, requires the extra memory and CPU capacity typically available on a tablet device.

Consistently with these findings, we developed a modular, Sesame-based architecture which can easily be scaled-up or down depending on the specific needs of a smart product and the resources available on a device, while ensuring a homogeneous architecture across heterogeneous devices. In particular we adapted the Sesame triple store for the Android system, and successfully applied it, through dedicated wrappers, on Android smartphones and tablet devices [19]. If required, this infrastructure component can be extended by enabling basic ontology reasoning or, when needed, a dedicated forward-chaining reasoner. In particular, following the results of the evaluation presented in [20], we have successfully used BaseVISor[11] to provide such inferencing support.

6 Evaluation

An evaluation of the SPO network of ontology can be found in [21]. Here, we focus instead on the evaluation of our technical architecture for realizing networks of smart products and we employ an approach based on the *ATAM methodology* [7], which is designed for evaluating architectures of collaborating agents. In particular, we consider the three following aspects[12], which are commonly used to evaluate distributed architectures –see also [22]:

(i) *Extensibility*: this aspect refers to the ability to add agents to the network which may implement additional capabilities.
(ii) *Reliability:* this aspect refers to the extent to which the failure of one or more agents in the network might affect the network as a whole.
(iii) *Security*: this aspect refers to the extent to which attacks on one or more elements of the network might expose the collective knowledge of the whole network.

In order to evaluate how our design choices impact on the properties considered above, we compare our architecture with the most obvious alternative solution, where ambiances are managed by a central 'facilitator' agent, which has the role of gathering the collective knowledge of the ambiance and providing matchmaking mechanisms to associate agents' needs to the capabilities of other agents. This solution differs from our approach because:

[10] http://www.openrdf.org/
[11] http://vistology.com/basevisor/basevisor.html
[12] Here we do not evaluate the *performance* of the architecture, as this is highly dependent on the implementation of each agent, as well as on the properties of the physical network used for communication between agents.

- It is *less distributed* –i.e., not *fully connected* [22]. In particular, the knowledge of each agent is not stored locally but at a central server.
- It is *reactive*, rather than proactive, as agents request features from other agents, rather than having smart products proactively contributing to shared tasks.

To evaluate the two alternative solutions, we consider four scenarios, which jointly cover the three properties mentioned earlier, and for each of them we assess qualitatively to what extent the two alternative architectures satisfy the relevant requirements. The scenarios have been chosen to illustrate generic situations that can be assessed independently from specific implementation details, and where the characteristics of the alternative architectures have a significant impact, as for example, when considering the security and privacy issues created by the introduction of a malevolent smart product.

Scenario 1 (Extensibility)*: Adding a smart product with capabilities and knowledge unknown to the network.*
In our framework, adding a smart product to an ambiance simply requires the corresponding device to join the peer-to-peer network. As knowledge is being held locally on the device and its capabilities used proactively to contribute to advertised tasks, there is no need to exchange any additional information. On the contrary, in a network where a facilitator maintains a directory of the available services/capabilities and aggregates the collective knowledge of the network, adding a smart product to an ambiance requires that the smart product registers its capabilities with the facilitator and constantly sends updates about newly generated knowledge. Besides the added complexity and communication overhead, this solution also requires that the facilitator either possesses a representation of any type of capability and knowledge that might be useful in a network, or is able to update its representation regularly, to comprise new capabilities and knowledge as they are introduced by smart products.

Extensibility in this sense is therefore one of the strong points of our proactive problem solving architecture: new smart products can contribute problem solving methods and knowledge directly as they join an ambiance, without the need for other agents in the network to have been explicitly programmed to manage such knowledge and capabilities.

Scenario 2 (Extensibility)*: Generating an ambiance by aggregating smart products.*
Because it is based on peer-to-peer communication, our framework supports the ability to create new ambiances by simply aggregating smart products in a network. In particular, this means that *ad-hoc* ambiances can be created 'on the fly', for example to connect the *Shopping Assistants* of a group of shoppers to allow them to share dynamically and proactively information in a supermarket –e.g., about what they are buying, the location of products in the shop, etc. Again, it is obvious that realizing this scenario would be far more complex if we were relying on a centralized, global knowledge architecture, where a selected smart product plays a controlling or facilitating role. The network would disappear as soon as this particular device becomes unavailable.

Another important aspect is that, because a smart product may inhabit multiple ambiances at the same time, it can transfer knowledge generated in one ambiance to another one –e.g., the *Shopping Assistant* inhabits both the supermarket and the kitchen ambiance, thus being able to share knowledge about special offers with other smart products in the kitchen ambiance. If a centralized approach were used, communication between

'facilitators' would be needed to achieve this result, creating an overhead in the best case and being simply unfeasible in most realistic scenarios.

Scenario 3 (Reliability): *One of the smart products in the network suddenly stops being operational.*
In our framework, as in any other, the impact of a particular agent's failure depends on the type of the agent. If this were a smart product with only an information-providing capability, such as the *Smart Fridge*, the impact would only be that the knowledge it contains would stop being available to the rest of the network, resulting in sub-optimal decision making. However, if the failing agent has more sophisticated capabilities (such as providing methods for meal planning), such problem solving capability would stop being available to the network, unless another smart product realizes a similar capability.

The situation is similar in networks relying on a facilitator agent, with some added complexity for the facilitator to handle situations in which registered features are requested, but are not available because of the corresponding device not being operational. A worst-case scenario in this type of network however is when the facilitator itself stops being operational, therefore invalidating all the capabilities and knowledge of all the smart products in the network.

Scenario 4 (Security): *A malevolent smart product is included in an ambiance.*
Here, we assume that a smart product has been created to join an ambiance in order to 'attack' it, meaning that its goal is to extract as much information as possible from the other agents in the network, or to disrupt collaborative problem solving. In the case of our framework, as all knowledge is localized in individual smart products and only shared when necessary for a particular task, the malevolent agent would need to be able to broadcast the right task and interpret properly the contributions from other agents to try and reconstruct the knowledge of other agents in the network. In addition, policies can be put in place on each device regarding the tasks they might contribute to, depending on the ambiance and the smart product that originated the task. For example, the *Shopping Assistant* might contribute to any task for which its capabilities are relevant in the kitchen ambiance, as it constitutes a trusted network, but may prefer to ignore tasks broadcast in the supermarket ambiance, as it has no reason to contribute there, and cannot verify whether these tasks are legitimate.

Of course, similar mechanisms can be put in place in the case of a facilitator-based network. Once again however, added complexity would be generated, as the facilitator would be required to implement mechanisms to consistently handle and manage policies for all smart products in the network. Obviously, the worst-case scenario here is when the problematic agent is the facilitator itself, since, as long as other agents can be tricked into joining its network, it would naturally collect all the knowledge of the other smart products in the ambiance. This is especially problematic in those cases where ad-hoc networks are formed (as discussed in Scenario 2), as one of the devices that might not be trustable will have to take the role of the facilitator, and could also potentially obtain information from the facilitators of other ambiances which have some devices in common with the one managed by the 'rogue' smart product.

7 Related Work

Proactive behaviour in artificial agents has been studied in the distributed AI community since the 70s and implementations of agents, which are able to exhibit proactivity, are often based on different variations of the *belief-desire-intention* framework (*BDI*) [23]. Problem-solving knowledge is usually decoupled into *goals* (what should be achieved) and *plans* (how to achieve it), in a similar way to the task-method decoupling in PSM research [11]. However, the notion of goal, while representing a necessary condition for achieving proactivity, does not *per se* reflect the behavioural patterns commonly associated with proactive behaviour –an agent can pursue a goal simply because it is asked to do so, as opposed to exhibiting proactivity, which requires that the agent actually takes the initiative in problem solving.

As already mentioned, an area of agent research, which specifically focuses on these issues, deals with the development of *user assistant agents* –e.g., see [2 8]. These studies consider proactivity as the capability of an agent "to anticipate needs, opportunities, and problems, and then act on its own initiative to address them" [2].

Our approach differs from the aforementioned ones in several respects. First, the use of standard Semantic Web representation makes it easier to integrate additional domain-specific information in our applications and take it into account during reasoning –e.g., as we do with information about food and recipes in our smart kitchen application [24]. Second, our approach involves exposing tasks as part of the shared context information, rather than by direct pairwise communication between agents. The reason for this is the need to deal with open environments, in which agents do not have prescribed roles. Thus, additional reasoning about whom to tell certain information is avoided and, while broadcasting may be considered in principle less efficient than direct agent to agent communication, in practice we have not found this to be an issue and we expect that even with reasonably large networks, our solution is unlikely to cause performance issues. In addition, we would also claim that our approach is more suitable for resource-constrained environments, as it uses more lightweight decision models than those used in most theories based on the BDI framework. In particular, we believe that unless we consider application-specific proactivity [2], where strong task models and learning mechanisms can be realized, our approach, which only requires task-based collaboration and does away with reasoning about other agents' beliefs and desires, provides a more 'agile' architecture to realise collaborative and proactive problem solving in networks of smart products.

In the ubiquitous computing area several approaches involving the use of ontologies and Semantic Web technologies have emerged, and some of them model the agent's activities [25 26]. However, these approaches pay less attention to the capabilities aspect, and the corresponding context broker implementations choose actions to perform using condition-action rules. The ontology developed in the CoDAMoS project [27] models user tasks and activities, as well as services provided by devices. Similarly, in the AMIGO project[12] process modelling is based on the notion of services, and a standard process representation ontology (OWL-S) is used to represent processes. These models allow matching tasks with device functionalities/services and representing process decomposition structures. Thus, decisions about when to contribute to a task can be made. However, they do not consider different capability modes, nor the participation of agents to multiple ambiances.

[12] http://www.hitech-projects.com/euprojects/amigo/

Several works have also targeted the integration of semantic data in small and mobile devices. In [28] an ad-hoc mechanism for storing and querying semantic web data on a mobile phone running iOS is presented, while [29] and [30] also describe mechanisms to integrate the use of semantic data within Android mobile phones. However, these solutions are restricted to the storage and manipulation of semantic data within 'closed' applications, while our approach provides the mechanisms needed to allow devices to exchange semantic data. Moreover, in contrast with our solution, which integrates Sesame with BaseVisor, none of these works consider the integration of inference engines. Tools such as μOR [31] exist for lightweight ontological reasoning on small devices, but do not integrate with common semantic data management infrastructures or with other types of reasoners.

8 Conclusions

In this paper we have provided an extensive overview of our work on smart products, in particular presenting a computational framework for realizing networks of smart products. The architecture is fully implemented and a demo of the smart kitchen application can be found at http://projects.kmi.open.ac.uk/smartproducts/demos/. For the future we plan to extend this work by investigating the augmentation of smart products with 'social intelligence', e.g., to enable them to act as 'social mediators' between users in open ambiances, such as a supermarket. In parallel we are also discussing with commercial partners the deployment of our architecture in large retail settings, where the ability for smart products to engage in proactive problem solving promises to open up new opportunities for customer-centric services.

References

1. Mühlhäuser, M. Smart Products: An Introduction. In Mühlhäuser et al. (eds), Constructing Ambient Intelligence. Comm. in Computer and Information Science, 11(4), 158-164, 2008.
2. Yorke-Smith, N., Saadati, S., Myers, K. and Morley, D. Like an Intuitive and Courteous Butler: A Proactive Personal Agent for Task Management. Eighth Int. Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'09), Budapest, Hungary, 2009.
3. Vildjiounaite, E., Kantorovitch, J., Kyllönen, V., Niskanen, I., et al. Designing Socially Acceptable Multimodal Interaction in Cooking Assistants. International Conference on Intelligent User Interfaces (IUI 2011), Palo Alto, 2011.
4. Beckerle, M., Martucci, L. A., Ries, S. Interactive Access Rule Learning: Generating Adapted Access Rule Sets. Second International Conference on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE 2010), Lisbon, Portugal, 2010.
5. Miche, M., Baumann, K., Golenzer, J., Brogle, M. A Simulation Model for Evaluating Distributed Storage Services for Smart Product Systems. 8th International ICST Conference on Mobile and Ubiquitous Systems, Copenhagen, Denmark, 2011.
6. Schreiber, A. T. Pragmatics of the Knowledge Level. Ph.D. Thesis, University of Amsterdam. Available at http://www.few.vu.nl/~guus/papers/Schreiber92c.pdf.
7. Woods, S. and Barbacci, M. Architectural evaluation of collaborative agent-based systems. Technical Report CMU/SEI-99-TR-025, SEI, Carnegie Mellon University, Pittsburgh, USA, 1999. Available at http://www.sei.cmu.edu/reports/99tr025.pdf.
8. Maes, P. Agents that reduce work and information overload. CACM 37(7), pp. 30-40, 1994.
9. Zhang, Y., Volz, R. A., Loerger, T. R., and Yen, J. A decision-theoretic approach for designing proactive communication in multi-agent teamwork. SAC 2004, pp. 64-71, 2004.

10. Baumgartner, N., Gottesheim, W., Mitsch, S., Retschitzegger, W., Schwinger, W. BeAware! - Situation awareness, the ontology-driven way. Data & Knowledge Engineering, 69, 2010.
11. Motta, E. Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving. IOS Press, Amsterdam, The Netherlands, 1999.
12. Chandrasekaran, B. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. IEEE Expert, 1(3), pp. 23-30, 1986
13. Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., et al. Knowledge Engineering and Management: The CommonKADS Methodology. MIT Press. 2000.
14. Murdock, J. W. and Goel, A. K. Meta-case-based reasoning: self-improvement through self-understanding. J. Exp. Theor. Artif. Intell. 20, 1 (March 2008), 1-36.
15. Domingue, J., Cabral, L., Galizia, S., Tanasescu, V., et al. IRS-III: A Broker-based Approach to Semantic Web Services, Journal of Web Semantics, 6, 2, pp. 109-132, Elsevier, 2008.
16. Aitenbichler, E., Kangasharju, J. and Mühlhäuser, M. MundoCore: A light-weight infrastructure for pervasive computing. Pervasive Mobile Computing, 2007. 3(4): p. 332-361.
17. Smith, R.G. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. IEEE Transactions on Computers 29(12), December 1980.
18. d'Aquin, M., Nikolov, A. and Motta, E. How much semantic data on small devices? 17th Int. Conference on Knowledge Engineering and Knowledge Management, EKAW 2010.
19. d'Aquin, M., Nikolov, A. and Motta, E. (2011) Building SPARQL-Enabled Applications with Android Devices. Demo at 10th International Semantic Web Conference (ISWC 2011).
20. Nikolov, A., Li, N., d'Aquin, M., Motta, E. Evaluating semantic data infrastructure components for small devices. Int. Workshop on Evaluation of Semantic Technologies (IWEST 2010) at 9th International Semantic Web Conference (ISWC2010), 2010.
21. Nikolov, A., d'Aquin, M., , Li, N., Lopez, V., et al. Evaluation of active components. SmartProducts Project Deliverable, D.2.5.1. Available at http://www.smartproducts-project.eu/media/stories/smartproducts/publications/SmartProducts_D2.5.1_Final.pdf.
22. Lee, S.K. and Hwang, C.S. Architecture modeling and evaluation for design of agent-based system. Journal of Systems and Software, 72 (2). pp. 195-208, 2004.
23. Rao, A.S., and George, M.P. Modeling rational agents within a BDI-architecture. 2nd Int. Conference on Principles of Knowledge Representation and Reasoning (KR'91), 1991.
24. Fernandez, M., Zang, Z., Lopez, V., Uren, V., Motta, E. Ontology Augmentation: Towards Healthy Meal Planning. 6th Int. Conf. on Knowledge Capture (K-CAP 2011) Banff, Canada.
25. Chen, H., Finin, T., and Joshi, A., The SOUPA Ontology for Pervasive Computing In: Ontologies for Agents: Theory and Experiences, pp. 233-258. Birkhäuser, 2005.
26. Wang, X. H., Zhang, D. Q., Gu, T., and Pung, H. K. Ontology Based Context Modeling and Reasoning using OWL. In 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops, pp. 18-22, 2004.
27. Preuveneers, D. et al. Towards an extensible context ontology for ambient intelligence. 2nd European Symposium on Ambient Intelligence, pp. 148-159, 2004.
28. Weiss, C., Bernstein, A., Boccuzzo, S.: i-MoCo: Mobile conference guide – storing and querying huge amounts of Semantic Web data on the iPhone/iPod Touch. Billion Triple Challenge ISWC 2008, Karlsruhe, Germany.
29. David, J., Euzenat, J.: Linked data from your pocket: The Android RDFContent-Provider. Demo at 9th International Semantic Web Conference (ISWC 2010).
30. Le-Phuoc, D., Parreira, J.X., Reynolds, V., Hauswirth, M.: Rdf on the go: A rdf storage and query processor for mobile devices. Demo at 9th International Semantic Web Conference (ISWC 2010).
31. Ali, S., Kiefer, S.: muOR - A Micro OWL DL Reasoner for Ambient Intelligent Devices. In: 4th International Conference on Advances in Grid and Pervasive Computing (GPC 2009).