Supporting Environment Configuration with Generic Role Assignment

Wolfgang Apolinarski, Marcus Handte, Pedro José Marrón
Networked Embedded Systems Group
University of Duisburg-Essen
Duisburg, Germany
{firstname.lastname}@uni-due.de

Abstract—**Due to the fact that distant objects are often less relevant to an application than objects in the proximity, middleware systems for pervasive computing typically exploit locality to improve efficiency. To do this, they configure the environment by introducing logical boundaries that reduce the number of interacting devices. Yet, in cases where applications require the interaction with distant objects, the boundaries become an artificial barrier that must be overcome by supplemental mechanisms. In this paper, we show how this problem can be avoided by using role assignment as a generic mechanism for environment configuration. To do this, we first derive the requirements for configuring a pervasive computing environment. We discuss how these requirements can be met by means of role assignment. To evaluate the approach, we present a prototypical implementation which we use to quantify the resulting overheads. The results indicate that role assignment enables a more flexible definition of boundaries at a low cost.**

Keywords-**Role Assignment, Adaptation, Configuration**

I. Introduction

Pervasive computing envisions seamless task support by means of applications executed on devices integrated into everyday objects. Thereby, pervasive applications try to minimize the user distraction by supporting tasks in an unobtrusive manner which requires a high degree of automation. Due to their integration, the devices encountered in pervasive systems are often resource-poor and specialized. Moreover, the devices usually interact with each other through wireless communication technologies and they may exhibit mobility. As a result, pervasive computing applications are usually distributed because they need to combine the specific capabilities of a number of devices. Furthermore, they must be adaptive to deal with the dynamics of the underlying networks.

To ease application development, existing middleware systems for pervasive computing can provide a diverse set of supportive mechanisms. At the lowest level, they can provide basic networking functionality. Beyond networking, the middleware systems can provide distributed object, service abstractions or component frameworks to simplify development, deployment and maintenance of applications. On top of that, they may provide intelligent automation, for example, to adapt applications to user preferences or to the available devices.

Intuitively, the resource utilization of many of these mechanisms is tightly tied to the number of devices that must be considered. As an example at the networking level, consider a mediator-based discovery scheme in which a central device collects all device information. Such a scheme may only work effectively in scenarios with a limited number of devices. At the automation level, the same holds true. For example, to automatically adapt a distributed application, it is often necessary to compute possible configurations. As the number of possible configurations increases with the number of devices, this kind of automation is hard to apply to large scale systems.

Since distant objects are often less relevant to an application than objects in the proximity, middleware systems typically exploit locality to improve performance. To do this, they introduce logical boundaries on the environment that reduce the number of devices. Although, the idea of exploiting locality is suitable in many scenarios, there are several cases in which applications may need to interact with distant devices. Some examples are scenarios that require access to resources in a remote smart environment or scenarios that require the remote collaboration of users in two separate smart environments.

In this paper, we show how this problem can be avoided by means of using role assignment for environment configuration. To do this, we first derive the requirements on environment configuration and thereafter, we describe how role assignment can be used as its basis. In order to evaluate the approach, we present a prototypical implementation which we use to quantify the resulting overheads. The evaluation indicates that role assignment enables a flexible definition of boundaries at a reasonable cost without introducing artificial barriers.

The remainder is structured as follows. Next, we revisit the problem of environment configuration and derive its requirements. In Section III, we describe the application of role assignment to environment configuration and in Section IV, we present a prototypical implementation. In Section V, we evaluate the approach and in Section VI, we describe related work. Finally, in Section VII, we conclude the paper.

II. Requirements

The main goal of environment configuration is to identify the set of devices of a pervasive computing system that may interact with each other. Intuitively, due to the dynamics of the underlying systems, the identification of these devices must be done continuously at runtime. From this goal, we can derive the following requirements on solutions that identify the set.

- *Configurability:* In environment configuration, the maximal set of devices is usually defined technically by means of connectivity. However, in order to maximize the performance, environment configuration typically strives for determining a minimum set. Yet, in many cases it is not feasible to clearly define the minimal set without running the risk of excluding devices that may be relevant for an application. In addition, a suitable definition may often be scenario-specific. As a consequence, a generic solution for environment configuration should be configurable in order to support the definition of effective boundaries in a scenario-specific way.
- *Flexibility:* Since distant objects are often less relevant to an application than objects in the proximity, existing middleware systems are exploiting locality to define the boundaries. Thereby, they consider a single characteristic of the context of a device such as a geographic location or physical proximity. Although, this approach has been proposed several times, it often does not result in minimal sets. In many scenarios, characteristics such as the device owner, for example, can be used more easily and result in smaller sets. Thus, instead of being fixed to a single context characteristic such as location, environment configuration should support the flexible definition of boundaries on various context characteristics.
- *Composability:* In order to avoid the introduction of artificial boundaries, environment configuration should support the on-demand extension of existing environments. However, in order to avoid side-effects between different applications and to allow the independent development of various definitions for the execution environment, the extension of the environment must be done in a controlled manner. To do this, environment configuration should support the composition of new environments by composing them from existing ones.
- *Efficiency:* A primary goal of environment configuration is to improve middleware and application performance by minimizing the set of devices that they must consider. As a consequence, environment configuration itself must be light-weight. This is specifically true, since the dynamics of pervasive systems require the continuous computation of the set. Furthermore, in order to support resource-poor devices, the mechanisms that are needed for configuration should exhibit a small size.

III. Approach

To provide a solution to environment configuration that fulfills the requirements described previously, we base our solution on the idea of role assignment. As a consequence, we first describe the overall idea of role assignment before we discuss how it can be applied to the problem of environment configuration.

A. Generic Role Assignment

As described in [1], the basis for generic role assignment is a set of devices that can communicate with each other.

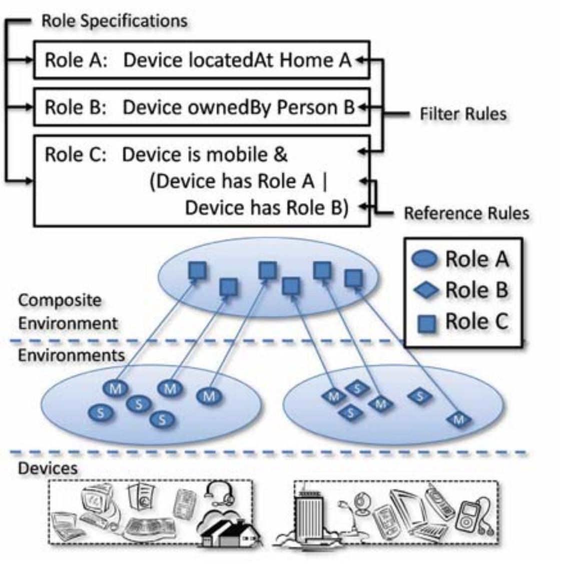
Role Specifications
Role A: Device locatedAt Home A
Role B: Device ownedBy Person B
Role C: Device is mobile & (Device has Role A | Device has Role B)
Filter Rules
Reference Rules
● Role A
◆ Role B
■ Role C
Composite Environment
Environments
Devices
M
S

Fig. 1. Using Role Assignment for Environment Configuration.

We assume that each device has some a priori knowledge about its context and that it is able to perceive parts of its context at runtime. Given that a lot of context such as the device type and owner for example are usually static and that more dynamic context such as the device's location can often be acquired automatically by means of built-in sensors or by retrieving sensor values from other devices, this assumption can be fulfilled by most mobile devices today. Furthermore, we assume that the context of a device is stored locally so that it can be accessed when needed.

Based on these assumptions, generic role assignment uses the device's context to assign roles. A *role* is essentially a tag that can be assigned to one or more devices. By definition, a role is assigned to any device as long as there are no further constraints that limit the assignment. To enable the automated computation of an assignment that reflects a particular goal, we introduce *rules*. Rules define contextual constraints on the assignment of roles to devices.

In [1], we identified 4 different classes of rules to support a broad range of configuration tasks. However, for environment configuration, we only require two classes of rules, which we call *filter rule* and *reference rule*. A filter rule simply constraints the set of devices to a set of devices that exhibits a particular context. An example for such a filter rule is to demand that all devices should be at a certain location. A reference rule references other role assignments. As an example for such a reference consider a rule that demands that a device must exhibit a particular role. Thus, by using reference rules, it is possible to assign roles hierarchically. The set of roles together with their corresponding rules form a role specification. To express complex logic, a single role may be constrained using several rules that are combined using the logical AND and OR operators. The logical NOT operator is not supported as this can easily lead to sets that require the evaluation of all globally connected devices. In practice, we did not find this to be problematic since it is usually possible to avoid NOT operators by an explicit enumeration of filters.

Given that the necessary contextual information is known to each device, we can automatically assign roles to the devices whose context satisfies the constraints specified by their rules. Alternatively, we may also empower a user to manually assign roles to support cases where the necessary context is not available or where automation is not desirable. However, in this paper, we focus on the automatic assignment, exclusively.

It is worth noting that a similar concept has also been proposed to configure sensor networks [2] and distributed robot systems [3]. However, the role specification and algorithms used in these works are specific to monitoring tasks and distributed robot coordination. As a consequence, the overall architecture and role specification language differ significantly. For example, the approach taken in [2] focuses primarily on network-related metrics whereas [3] applies utility functions to achieve a targeted coverage.

B. Application to Environment Configuration

To apply generic role assignment to environment configuration, we can use role specifications to define the boundaries. The actual assignment of a single role can then be used to define the set of devices. Thereby, we may reason about roles from the perspective of the device, i.e. whether the device has a certain role, or from the perspective of the overall system, i.e. which set of devices has a certain role. Thus, we can identify whether a particular device belongs to the environment and we can identify the total set of devices that form the environment.

Figure 1 shows an example for this. To specify the typical boundaries of a smart space, a developer can create a role specification that assigns Role A to all devices whose location is known to be inside Home A. To do this, the developer creates a filter rule for the location and attaches it to role A. After the assignment, the devices within the home can be identified by the role. Intuitively, in order to cope with changes, the assignment process must be performed at regular intervals. Similarly, in order to specify the boundaries on the basis of device ownership, a developer can specify a filter rule that constraints the set of devices to a particular person (Role B). In order to combine these sets of devices, two reference rules can be used to reference the roles A and B. Using Boolean operators it is possible to further restrict the set of devices, e.g. to only select mobile ones. As we discuss later on, the resulting role assignments can then be used independently from their definition to optimize middleware functions.

IV. Implementation

To evaluate the approach, we have implemented a prototypical role assignment system. In the following, we first describe the architectural components. Thereafter, we describe how they interact. Finally, we describe some example mechanisms that use an assignment for optimization.

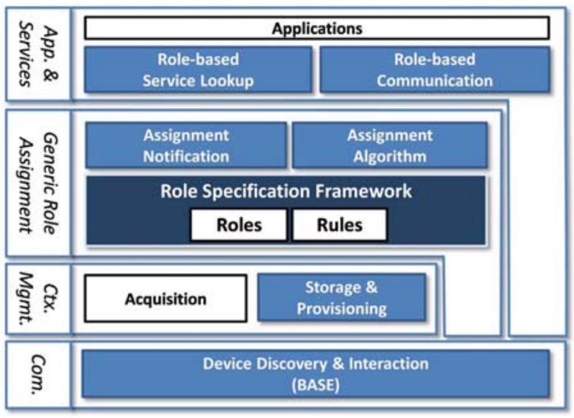

Fig. 2. Generic Role Assignment System Architecture.

A. Architecture

The individual layers and the high-level building blocks of our generic role assignment system are depicted in Figure 2. Conceptually, the four main layers are communication, context management, role assignment and services that use the assignment as environment definition.

To enable communication between devices, we rely on BASE [4], an existing communication middleware for pervasive systems. BASE provides the basic communication functionality such as support for device discovery and interaction. On top of that, BASE provides a basic service model that we use to implement the remaining layers of the role assignment system. Thereby, every building block is implemented as a well-known service that can be accessed locally and remotely.

To automate role assignment, the system needs to be able to automatically capture context information. Due to the differences in sensor APIs for various devices, the acquisition must usually be done in a device-dependent manner. Additionally, the context management layer is responsible for abstracting from the details of gathering context information by providing a uniform query interface. To represent context information, we are using RDF [5] which enables data modeling and reasoning on the basis of standard ontology languages such as OWL. In order to query the RDF data, we use SPARQL [6] basic graph matching patterns which we extend with non-standard geo-spatial extensions for range and nearest-neighbor queries. This allows us to express location-based queries which are often useful for environment configuration. Consequently, filter rules are formulated as SPARQL queries whose successful evaluation determines whether a device matches the filter. In order to support context provisioning on resource-poor devices, we have implemented two alternative context services for different classes of devices. On resource-rich devices, we use JENA and ARQ to store context and to evaluate queries. On resource-poor devices, such as mobile phones or Sun SPOTs, we use a custom implementation that stores all context information in-memory and implements a large subset of the SPARQL language with limited reasoning capabilities over a set of statically compiled ontologies.

On top, the generic role assignment layer provides the functionality to define role specifications using roles and rules. Once a role specification is passed to the role assignment layer, it can automatically perform the assignment using context information. To do this, the layer provides an assignment service that computes an assignment. Once the assignment has been computed, the roles need to be distributed to the devices. This enables them to determine whether they exhibit a certain role. To perform this distribution in an application-independent manner, the role assignment layer includes a notification service which is notified by the assignment service whenever a local assignment changes. Note that the assignment service is not needed by each device. Instead, it is only necessary on those devices that are actually computing an assignment. Thus, to minimize the resource consumption, it is possible to deploy only the notification service.

At the service layer, other services and applications may use the role assignments to optimize their mechanisms. Thereby, they can use the local notification service to react to changes of roles. Alternatively, they can query the assignment service in order to retrieve the current assignment. The former reflects the per device view, the latter reflects the system view.

B. Interaction

To clarify the architecture, we describe the runtime interaction of its components in the following. As explained earlier, each device is equipped with an instance of BASE and the additional services that form the generic role assignment system. To configure an environment or an application, a middleware service may start a role specification by sending it to a device equipped with an assignment service.

Since multiple role specifications may use the same role identifiers, the role assignment service first creates a globally unique id for the specification. This enables the unique identification of individual roles which is required to reference a particular role. To do this, the role assignment service concatenates the BASE device id with a locally unique id.

Once the id has been assigned, the assignment service analyses the specification to determine whether the role specification references some other role specification by means of reference rules. If the role specification does not contain reference rules, the assignment service creates a list of all SPARQL queries that represent the filter rules. Thereafter, it sends a single batch query to all connected devices. Once the list of responses is returned, the role assignment service evaluates the Boolean expression over the rules and computes the assignment.

If the role specification contains reference rules, the assignment service forwards the specification to the assignment service that is executing the referenced role specification. If a role contains multiple references, the specification is forwarded to each referenced assignment service. The assignment service that receives the specification will then execute it locally. Thereby, it considers only those reference rules that reference local assignments. The other rules are simply ignored. After the assignment has been computed at the referenced assignment service, a list of candidate assignments is returned

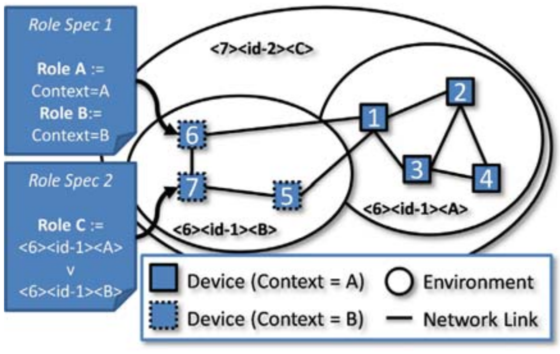

Fig. 3. Generic Role Assignment Example

to the original assignment component. There, the candidate assignments are transformed into final assignments. To do this, the assignment component may have to intersect or unify the candidate sets in order to compute the result in cases where multiple references are concatenated using a conjunction or a disjunction.

Once the final role assignment has been determined, the role assignment service calls the notification service on each device that receives at least one role. Thereby, the service transmits all assigned roles. Applications may register local listeners at their notification service to receive changes to assignments.

An example for this process is depicted in Figure 3. The figure shows 7 devices that execute two role specifications. The first role specification defines two environments using the roles A and B. Both roles solely rely on filter rules in order to define the sets of devices. In order to keep the figure simple, we refrain from using SPARQL syntax, instead we simply assume that role A requires context A and role B requires context B. Once the role specification is started at device 6, the device assigns a unique id, i.e. <6><id-1>. Thus, the roles can be identified by concatenating the role specification id with the role name, i.e. <6><id-1><A> or <6><id-1><B>. Since there are only filter rules, the assignment component queries the context of the connected devices and computes the assignment according to the rules. Finally, the assignment component notifies all devices that received a particular role.

The second role specification in the example refers to the first specification to define an environment using role C that consists of all devices that have role A or B. When the role specification is started at device 7, the unique id is generated and the role specification is analyzed. Since the role specification contains reference rules, the role specification is forwarded to the devices that are managing the referenced specification. In this example, this is done by device 6. To determine the managing device, the device 7 can simply use the BASE id that is embedded in the reference. Device 6 then computes the candidate set consisting of devices with role A and role B and returns it to device 7 which performs the final assignment. In this example, the candidate set and the final set are identical. However, if several specifications on multiple devices are referenced, it may be impossible to determine the

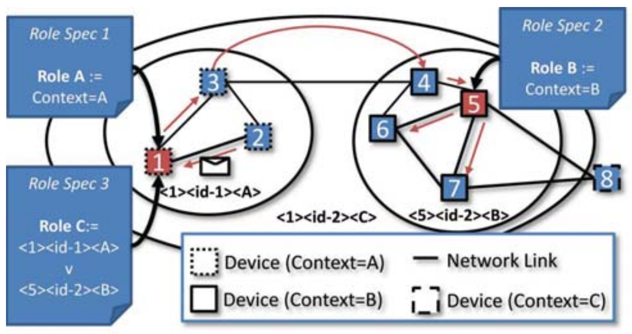

Fig. 4. Role-based Communication

set locally on the referenced devices. Once the set has been computed, device 7 notifies all relevant devices.

C. Integration

To validate the architecture of the role assignment system with respect to its interfaces, we modified the BASE service registry so that it benefits from the environment configuration. In addition, we have implemented a BASE communication plug-in that provides environment-based communication. In the following, we briefly outline the implementation.

To enable the spontaneous interaction of devices, BASE not only supports device discovery and interaction but it also provides a simple service abstraction. In order to find local and remote services, BASE provides a service registry. To support the dynamics of pervasive systems, the BASE service registry uses a reactive federation scheme. Each BASE-enabled device is equipped with a local registry that can be accessed locally as well as remotely. In order to export a service, an application simply calls an export function on the local registry which stores the associated service information. To search for available services, an application can call a search function locally. Internally, the search is then automatically distributed across all devices in order to return the complete set of services.

To improve the efficiency of the federation scheme, we have extended the registry to support the search within a particular environment. To do this, an application developer can define an environment using a role specification. Later on, the developer can search within the environment by sending a query and an associated globally unique role identifier to the local registry. Internally, the registry will then first contact the assignment service to retrieve the devices that exhibit the role and later on, it will only forward the queries to these devices. Thus, we speed up the search by minimizing the set of devices.

In addition to this, we have implemented a BASE communication plug-in that provides environment-based communication. Similar to the service registry, the communication plug-in restricts the distribution of a particular message to an environment that is defined by a role assignment. To distribute the load of message forwarding, the plug-in uses the hierarchy that is created by reference rules for distribution. When a device receives a role, it may use the role to join a group communication channel using the BASE plug-in. If a message must be transmitted, the plug-in simply forwards the message to the device that performs the assignment. This device then forwards it to other devices, either directly - if it has performed the assignment - or indirectly - if it uses another device to compute candidate sets.

An example for this is depicted in Figure 4. If device 2 sends a message using the channel defined by role C, it forwards the message to device 1, since this device is responsible for performing the assignment. Device 1, in turn, uses device 5 to compute parts of the assignment and thus, it forwards the message to this device. Furthermore, device 1 distributes the message to all devices with the role A, since it has performed the assignment for this role. If the message arrives at device 5, the device distributes the message to all devices with role B, since it is responsible for assigning this role.

V. Evaluation

In this section, we evaluate the approach. To do this, we first discuss the requirements on configurability, flexibility and composability before we determine efficiency experimentally.

A. Discussion

As discussed in Section II, approaches that support environment configuration should be configurable, flexible and composable to be applicable to a broad range of scenarios. In the following, we briefly discuss why and how generic role assignment fulfills these requirements.

- *Configurability:* By design, environment configuration that is implemented using generic role assignment can be flexibly configured to meet the needs of the applications. To do this, an application developer may specify arbitrary filter and reference rules that can be evaluated automatically at runtime. Thereby, the developer may start and stop a number of role specifications on-demand in order to enable the definition of sets that result in an optimal middleware and application performance.
- *Flexibility:* Environment configuration with generic role assignment is not primarily based on location. Instead, it enables developers to define boundaries using properties of the device context. Clearly, in order to use a property in a role specification, it must be available on the relevant devices. However, when looking at the increasing number of sensors that are deployed in current smart devices, it is conceivable that many devices will be able to perceive a large part of their context. As a result, role assignment increases the flexibility of environment configuration when contrasted with the locality-based approaches.
- *Composability:* Generic role assignment is not limited to a single role specification. Instead, multiple specifications may be developed independently and executed simultaneously. The support for reference rules within role specifications enables the hierarchical composition of environments. By supporting the hierarchical composability, generic role assignment can be used to dynamically extend existing environments in a controlled fashion.

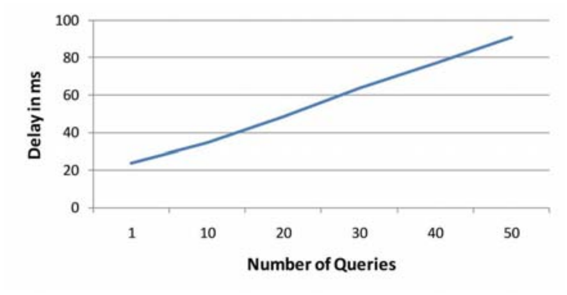

Fig. 5. Latency with Varying Number of Queries

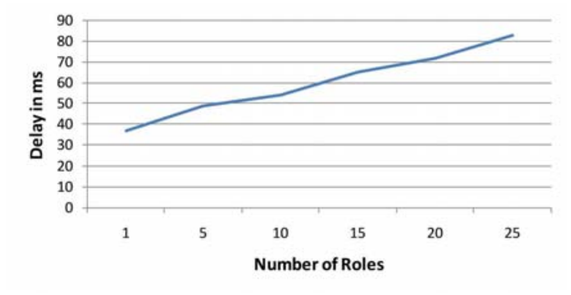

Fig. 6. Latency with Varying Number of Roles

B. *Experiments*

In the following, we take a look at efficiency. To validate that the system can be used on resource-poor devices, we measured the binary size of the additional Java code. Using our minimal context service implementation that does not use JENA and ARQ, the services require an additional memory space of 140KB. However, this space can be reduced to 85KB if a system does not have to perform role assignment. Thus, the role assignment system supports a broad range of devices including phones or embedded devices such as Sun SPOTs.

Furthermore, we performed a number of experiments using the previously presented implementation. For all experiments, we have used the following hard- and software configuration consistently. We use an off-the-shelf Asus EEE PC T91 (Intel Atom Z520 1.33 GHz CPU, 1 GB RAM) running Windows XP and Sun JRE1.6 to perform assignments and we connect it to a varying number of devices. For this, we use HTC Tattoos (Qualcomm MSM7225 528 MHz CPU, 256MB RAM) running Android 1.6. To connect the devices, we use an IEEE 802.11g wireless network hosted by a Netgear WNR3500L access point which is used exclusively for the experiments.

The performance of role assignment depends on the contents of the role specification. The two primary influential factors are thereby the number of queries in filter rules and the number of roles that shall be distributed. In order to measure the impact of these factors, we performed two experiments which vary them systematically. In both experiments, we execute the role specifications 200 times with one HTC Tattoo and we measure the delay experienced for performing a single assignment.

Figure 5 shows the results of the first experiment for which we create a role specification consisting of one role that contains a varying number of queries for one RDF triple (1-50). Figure 5 indicates a linear growth of the average latency for role assignment starting from approximately 25 ms when one query is attached up to approximately 90 ms when 50 queries are present. Due to the batch processing for query execution, this increase can be explained by the increased effort for serialization, transmission and remote execution.

To evaluate the effects of an increasing number of roles, we use the same setup with one device but we change the role specification to contain a varying number of roles (1-25). Thereby, each role queries one triple. Similar to Figure 5, we can observe a linear increase in latency for performing the role assignment as depicted in Figure 6. However, the increase is approximately twice as steep with a absolute latency of approximately 90 ms when distributing 25 roles. Similar to the experiment that varies queries, we can attribute this to the increased effort for serialization, transmission and execution. The reason for the higher increase results from the fact that each role also contains one query.

In summary, we conclude from these two experiments that increasing the number of roles or the number of queries result both in a linear increase of the latency experienced in role assignment. Thereby, the absolute values of less than 100 ms clearly indicate the suitability for comparatively resource-poor devices. In order to measure the effects of an increasing number of devices, we have performed two additional experiments. In the first one, we measure the overhead for evaluating filter rules. In the second one, we measure the overhead for hierarchical assignment using reference rules. In each experiment, we measure 200 role assignments and compute the average role assignment latency.

Figure 7 shows the effects on latency when increasing the number of devices in an assignment that uses filter rules. The role specification in this experiment consists of one role with one filter rule that queries a single context property. As indicated in Figure 7, increasing the number of devices also increases the latency. However, when comparing the absolute values it becomes apparent that an increase in the number of devices only causes a comparatively marginal increase in the overall latency. For example, the role assignment with 2 devices is less than twice more expensive than with 1 device. The reason for this can be attributed to the fact that the filter rules contained in the role specification can be executed on the devices in parallel. However, in practice the achievable gain from this parallelism also depends on the amount of data that is transferred. Thus, for an increasing number of devices, the increase in latency would eventually approximate direct proportionality due to network saturation.

As show in Figure 8, such effects are not present when a specification contains only reference rules. In the experiment depicted in this figure, we evaluate the latency for assigning a role to a varying number of devices on the basis of an existing role. The reference rules used in this experiment reference two role specifications that are running in the system. In order to show the actual effort for evaluating the reference rule,

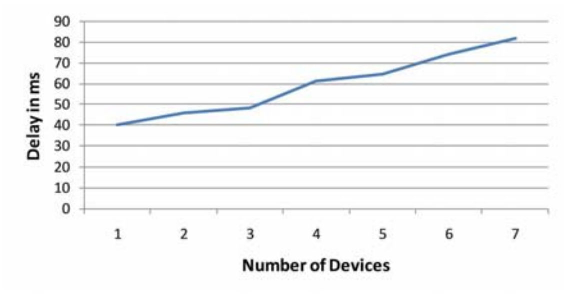

Fig. 7. Latency for Filter Rules with Varying Number of Devices

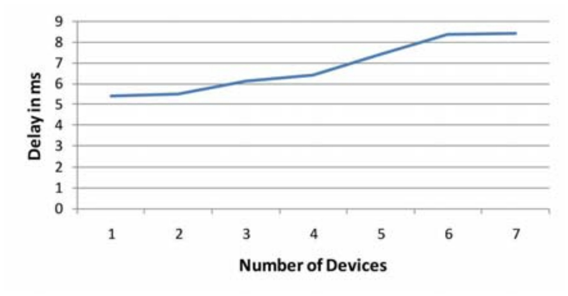

Fig. 8. Latency for Reference Rules with Varying Number of Devices

we do not measure the delay for device notification. Due to the fact that the computation of the assignment can be done completely locally on the assigning device, the overall time for assignment stays well below 10 ms for all experiments. This clearly indicates the usefulness of reference rules and it also demonstrates the effectiveness of hierarchical composition, especially when forming a group from existing groups.

In most settings, however, we would expect to see a combination of reference and filter rules which would prevent a purely local evaluation. Yet, in these cases, our implementation restricts the number of devices that must evaluate the filter rules to those that already posses the desired set of roles. Thus, instead of contacting all nearby devices, the device performing the role assignment only has to contact a subset. Given the low effort of reference rules, this approach is often beneficial.

Note that we can simply compute the overall effort for such more realistic settings from the synthetic measurements. Since the delays shown in Figure 8 do not contain the delay caused by notifications, we can directly sum up the efforts for hierarchical assignment shown in Figure 8 with the effort shown in Figure 7. As a concrete example consider the following. In the PECES [7] European research project, role assignment is used as a basic abstraction to form a smart space by dynamically distributing three roles for *member*, *gateway* and *coordinator* devices. The coordinator devices are responsible for providing centralized services such as role-based group communication and service discovery. Gateway devices are responsible for connecting the devices of a smart space with other smart spaces and member devices may provide and use services present in the smart space. Within the scenarios considered by PECES the number of devices contained within a single space typically ranges between 3 devices (for a simple in-car smart space) up to 10 devices (for an in-house smart space) which often triples once different smart spaces begin to interact.

In a medium-sized smart space that consists of 6 devices where roles are distributed using one filter rule each, the total overhead introduced by role assignment can be estimated as follows: The total time to set up the environment is computed from the execution of the three queries (26 ms), the distribution of 8 roles (6 member, 1 gateway and 1 coordinator, 52 ms) and the role assignment latency for 6 devices (74 ms). This adds up to 152 ms for each setup. To detect changes, the role assignment is performed at regular intervals of 30 seconds. Thus, the overhead introduced by role assignment in this case is well below 1 percent. We therefore conclude that the role assignment approach also fulfills the efficiency requirement.

VI. Related Work

Most existing middleware systems for pervasive computing exploit locality to improve their performance. To do this, they configure the execution environment by introducing logical boundaries that reduce the number of interacting devices. Usually, these boundaries are defined on the basis of proximity or location depending on the underlying system model.

Middleware systems that support smart spaces such as IROS [8], Gaia [9], Easy Living [10], Aura [11] or Oxygen [12] are usually bound to a specific geographic location. This location may represent a building such as a home or a work place [10], [12] or a single room such as a meeting room or an office [9], [8], [11], [13]. Within this area, a coordinating server is responsible for providing additional services such as shared persistent storage, context management or application configuration, for example. Thereby, the server is responsible for dynamically handling the mobile devices that enter or leave the area. However, without additional mechanisms, these systems cannot cross the boundaries of the area.

Middleware systems that support smart peers such as BASE [4] and PCOM [14] or MundoCore [15], for instance, are usually relying on different proximity metrics. With these metrics the define the boundaries around each device, for example, as the set of devices in n-hop neighbourhood [16]. Similarly, concepts such as abstract regions [17] and scenes [18] use location as reference point to form dynamic environments around it. For defining the boundaries, these approaches are limited to topological or geographical regions. Logical neighbourhood [19] and hood [20] are two approaches that restrict the scope to the physical (i.e. 1-hop) neighbourhood.

To allow programmers addressing the specific regions in a network, SpatialView [21] provides a programmable abstraction over different properties of the underlying network. Similarly, in Regiment [22], the programmer views the complete network as geographical or topological streams and he can manipulate these streams to address a region. The above approaches are useful in defining the pervasive environment in immediate vicinity. EnviroTrack [23], targets towards application tracking, goes beyond physical closeness and focuses on data centric communication between entities with similar context. However, these approaches do not support composition.

From the perspective of composability, the presented work is close to UbicKids [24] and Superspace [25]. UbicKids provides mechanisms for enabling cross pervasive environment communication by exposing services to the existing UbicKids pervasive environments. Superspaces utilize active spaces [9] that are by themselves based on geographic locations. Thus, they are an additional mechanism that has been applied to extend the boundaries of the underlying smart space.

Environment configuration that is based on generic role assignment, as described in this paper, can naturally support these cases without additional mechanisms. In addition, it can support other notions of context-dependent environments that cannot be supported by existing approaches and systems.

VII. Conclusion

Since distant objects are often less relevant than objects in the proximity, pervasive computing middleware systems typically exploit locality to improve efficiency. Thereby, they introduce artificial boundaries that may become a hindrance.

In this paper, we have shown how generic role assignment can be used as basis for environment configuration. Furthermore, we have presented a prototypical role assignment system. The overhead induced by our implementation indicates that role assignment can be used effectively to exploit locality. At the same time, generic role assignment allows the expansion of the boundaries in a hierarchically structured way which prevents the introduction of artificial barriers at a low cost.

In the PECES research project, we are extending the role assignment system to support other types of configuration. Currently, we are focusing on support for the configuration of access rights. For this, we are adding security primitives on top of our pervasive authentication framework [26].

Acknowledgments

This work has been partially supported by CONET (Cooperating Objects Network of Excellence) and PECES (PErvasive Computing in Embedded Systems), both funded by the European Commission under FP7 with contract numbers FP7-2007-2-224053 and FP7-224342-ICT-2007-2.

References

[1] M. Haroon, M. Handte, and P. J. Marron, “Generic role assignment: A uniform middleware abstraction for configuration of pervasive systems,” *PerWare Workshop at the 7th Annual IEEE International Conference on Pervasive Computing and Communications*, 2009.

[2] C. Frank and K. Römer, “Algorithms for generic role assignment in wireless sensor networks,” in *SenSys '05: 3rd international conference on Embedded networked sensor systems*. NY, USA: ACM, 2005, pp. 230–242.

[3] L. Iocchi, D. Nardi, M. Piaggio, and A. Sgorbissa, “Distributed coordination in heterogeneous multi-robot systems,” *Auton. Robots*, vol. 15, pp. 155–168, September 2003.

[4] M. Handte, C. Becker, and G. Schiele, “Experiences - extensibility and minimalism in BASE,” in *Workshop on System Support for Ubiquitous Computing (UbiSys) at Ubicomp*, 2003.

[5] W. W. W. Consortium, “Resource description framework.” [Online]. Available: http://www.w3.org/standards/techs/rdf

[6] World Wide Web Consortium, “Sparql query language for rdf.” [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[7] P. Consortium, “Peces european research project website,” 2010. [Online]. Available: http://www.ict-peces.eu

[8] S. R. Ponnekanti, B. Johanson, E. Kiciman, and A. Fox, “Portability, extensibility and robustness in iros,” in *PERCOM '03: 1st IEEE International Conference on Pervasive Computing and Communications*. Washington, DC, USA: IEEE Computer Society, 2003, p. 11.

[9] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, “Gaia: a middleware platform for active spaces,” *Mobile Comput. Commun. Rev.*, vol. 6, no. 4, pp. 65–67, 2002.

[10] B. B. Brian, B. Meyers, J. Krumm, A. Kern, and S. Shafer, “Easyliving: Technologies for intelligent environments.” Springer-Verlag, 2000, pp. 12–29.

[11] D. Garlan, D. P. Siewiorek, and P. Steenkiste, “Project aura: Toward distraction-free pervasive computing,” *IEEE Pervasive Computing*, vol. 1, pp. 22–31, 2002.

[12] L. Rudolph, “Project oxygen: Pervasive, human-centric computing - an initial experience,” in *CAiSE '01: 13th International Conference on Advanced Information Systems Engineering*. London, UK: Springer-Verlag, 2001, pp. 1–12.

[13] S. S. Yau, S. K. S. Gupta, E. K. S. Gupta, F. Karim, S. I. Ahamed, Y. Wang, and B. Wang, “Smart classroom: Enhancing collaborative learning using pervasive computing technology,” in *In ASEE 2003 Annual Conference and Exposition*, 2003, pp. 13 633–13 642.

[14] C. Becker, M. Handte, G. Schiele, and K. Rothermel, “Pcom - a component system for pervasive computing,” in *2nd IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*. Washington, DC, USA: IEEE Computer Society, 2004, p. 67.

[15] E. Aitenbichler, J. Kangasharju, and M. Mühlhäuser, “Mundocore: A light-weight infrastructure for pervasive computing,” *Pervasive Mob. Comput.*, vol. 3, no. 4, pp. 332–361, 2007.

[16] G.-C. Roman, C. Julien, and Q. Huang, “Network abstractions for context-aware mobile computing,” in *ICSE '02: 24th International Conference on Software Engineering*, NY, USA, 2002, pp. 363–373.

[17] M. Welsh and G. Mainland, “Programming sensor networks using abstract regions,” in *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, Berkeley, CA, USA, 2004, pp. 3–3.

[18] S. Kabadayi and C. Julien, “A local data abstraction and communication paradigm for pervasive computing,” in *5th IEEE International Conference on Pervasive Computing and Communications (PerCom'07)*, 2007, pp. 57–68.

[19] L. Mottola and G. P. Picco, “Using logical neighborhoods to enable scoping in wireless sensor networks,” in *MDS '06: 3rd international Middleware doctoral symposium*. NY, USA: ACM, 2006, p. 6.

[20] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler, “Hood: a neighborhood abstraction for sensor networks,” in *MobiSys '04: 2nd international conference on Mobile systems, applications, and services*. NY, USA: ACM, 2004, pp. 99–110.

[21] Y. Ni, U. Kremer, and L. Iftode, “Spatial views: Space-aware programming for networks of embedded systems,” in *16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2003*, 2003.

[22] R. Newton and M. Welsh, “Region streams: functional macroprogramming for sensor networks,” in *DMSN '04: 1st international workshop on Data management for sensor networks*, NY, USA, 2004, pp. 78–87.

[23] T. Abdelzaher, B. Blum, Q. Cao, Y. Chen, D. Evans, J. George, S. George, L. Gu, T. He, S. Krishnamurthy, L. Luo, S. Son, J. Stankovic, R. Stoleru, and A. Wood, “Envirotrack: Towards an environmental computing paradigm for distributed sensor networks,” in *ICDCS '04: 24th International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 582–589.

[24] J. Ma, L. T. Yang, B. O. Apduhan, R. Huang, L. Barolli, M. Takizawa, and T. K. Shih, “A walkthrough from smart spaces to smart hyperspaces towards a smart world with ubiquitous intelligence,” in *ICPADS '05: 11th International Conference on Parallel and Distributed Systems*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 370–376.

[25] J. Al-muhtadi, S. Chetan, and R. Campbell, “Super spaces: A middleware for large-scale pervasive computing environments, perware 04,” in *IEEE International Workshop on Pervasive Computing and Communications*, 2004, pp. 198–202.

[26] W. Apolinarski, M. Handte, and P. J. Marrón, “A secure context distribution framework for peer-based pervasive systems,” in *PerWare Workshop at the 8th Annual IEEE International Conference on Pervasive Computing and Communications*, March 2010.