
Ran Zhao, Neil Speirs
School of Computing Science, Newcastle University, Newcastle NE1 7RU, United Kingdom
{ran.zhao1, neil.speirs}@ncl.ac.uk

Kirusnapillai Selvarajah
Department of Electrical and Electronic Engineering, The University of Nottingham, Nottingham NE7 2RD, United Kingdom
kirusnapillai.selvarajah@nottingham.ac.uk

Abstract — The main objective of the PECES project is the development of system software to enable the communication among heterogeneous devices across multiple smart spaces, breaking the traditional barrier of “smart islands” where only the services offered in a nearby spatial area can be used easily. PECES development tools help the application developer to build and test the PECES middleware based applications. This paper presents a set of tools, namely Peces Project, Peces Device Definition, Peces Ontology Instantiation, Peces Service Definition, Peces Role Specification Definition, Peces Hierarchical Role Specification Definition, Peces Event Editor, Peces Event Diagram and Peces Testing which enable application developers to build, model and test the PECES middleware based smart space application using the novel concepts such as role assignment, context ontologies and security.

Index Terms—smart space, middleware, pervasive computing, wireless networking, context ontologies, modelling, testing, dynamic addressing, eclipse plugins, communication gateway, registry interface.

I. PECES PROJECT

The objective of the PECES project [1] is the creation of a comprehensive software layer to enable the seamless cooperation of embedded devices across various smart spaces on a global scale in a context dependent, secure and trustworthy manner. The increasing number of devices that are invisibly embedded into our surrounding environment as well as the proliferation of wireless communication and sensing technologies are the basis for visions like ambient intelligence, ubiquitous and pervasive computing. The benefits of these visions and their undeniable impact on the economy and society have led to a number of research and development efforts. These include various European projects such as EMMA [2], [3] that develop specialized middleware abstractions for different application areas such as automotive and traffic control systems or home automation. Middleware for pervasive environments primarily manages the stationary infrastructure in the environment. Usually, this infrastructure consists of stationary devices deployed within a predefined physical location such as meeting room or car parking. There are several other middleware systems have been developed for this purpose such as Aura [19], Gaia [12], [13] and IROS [20]. Reliable service management framework is proposed in [21] by formally defining a message-oriented service application model and protocols that facilitate autonomous composition, failure detection and recovery of services.

These efforts have enabled smart spaces that integrate embedded devices in such a way that they interact with a user as a coherent system. However, they fall short of addressing the cooperation of devices across different environments. This results in isolated “islands of integration” with clearly defined boundaries such as the smart home or office. For many future applications, the integration of embedded systems from multiple smart spaces is a primary key to providing a truly seamless user experience. Nomadic users that move through different environments will need to access information provided by systems embedded in their surroundings as well as systems embedded in other smart spaces. The PECES project is committed to developing the technological basis to enable the global cooperation of embedded devices residing in different smart spaces in a context-dependent, secure, and trustworthy manner. The most innovative features of the PECES middleware are to enable the communication among heterogeneous devices across the different smart spaces using dynamic addressing, security and context ontologies.

The PECES project has the following scientific and technical objectives:

1. Development of a flexible ontology to capture the context of cooperating objects and to specify groups of cooperating objects in an abstract manner.
2. Development of a middleware – i.e. a set of application independent services that enable the dynamic
1. Communication Gateway

Due to the heterogeneity of devices and communication technologies, it is not safe to assume all future devices will be equipped with the same set of communication technologies. As an example consider that a sensor node might only be equipped with ZigBee (based on IEEE 802.15.4) but not with Bluetooth in order enable energy-efficient communication. Thus, in order to enable a Bluetooth device to communicate with such sensor nodes, it is necessary to use a device that is equipped with both technologies as a local gateway. Similarly, due to the associated costs and other factors, not all devices will have a direct connection to a global interconnection network like the Internet. In order to enable the communication between devices that are not directly connected to the Internet, it is necessary to enable some devices to act as remote gateways for others.

PECES middleware support local gateways as well as remote gateways. The main difference between these two types of gateways is that the local gateway locally shares the required knowledge. In the remote case, the knowledge sharing should be restricted to a minimum in order to avoid the costly distribution of frequently changing information. The remote gateways need to be realized differently in that they require an external entity to distribute the information that is distributed by means of device discovery in the local case. This information will be distributed by means of the registry that is specified in the PECES Communication Mechanism and Registry Interface Specification [8].

2. Generic Role Assignment

Due to the continuous changes in context and due to the mobility of devices the underlying systems can be highly dynamic and the network topology can change frequently in the pervasive computing environments. So that it is vital to enable pervasive computing applications such as PECES prototype applications to adapt to the continuous changes in context and device availability. The responsibility for adaptation can be shifted between different entities. In cases where changes are infrequent, a user may manually configure and adapt the system. However, if changes are frequent, manual configuration and adaptation are clearly not a viable approach as they conflict with the goal of distraction free support for tasks. In order to mitigate this, the adaptation can be automated through the application. This approach relieves the user from performing manual adaptation but it complicates the development of applications and it may result in inefficiencies in cases where multiple applications implement and use similar adaptation mechanisms. As a result, the PECES middleware is aiming at automating the initial configuration and the continuous adaptation to changes in order to shield the user and the application developer from the accompanied complications.

In order to be suitable for a broad range of different systems and in order to minimize the utilization of resources that are required for automation, PECES middleware provides configuration and adaptation support by means of a uniform abstraction. To create a
uniform abstraction that is suitable for a broad range of
different configuration tasks, it is necessary to introduce a
clear separation between the result of a configuration, the
computations that need to be done to produce it and the
utilization of this result. This enables the reuse of the
same basic mechanisms for different tasks. Generic role
assignment provides such a uniform abstraction. More
detailed information about the role assignment concepts
can be found in [7].

A role can be 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 of a configuration task, generic
role assignment introduces rules. Rules define contextual
constraints on the assignment of roles to devices. The
simplest form of contextual constraint that is generally
useful for all configuration tasks is a simple filter. An
element of such a filter is to demand that all devices
should be at a certain location. Another form of
contextual constraint that is particularly relevant for
PECES middleware are so called reference rules.
Reference rules refer to a set of devices that has been
assigned a particular role.

The set of rules together with their corresponding roles
form a role specification. Given that the necessary
contextual information can be captured by sensors or
other types of information sources, one can use an
algorithm to automatically assign roles to the devices
whose context satisfies the constraints specified by their
rules.

3. Smart Space Concept

A smart space can be defined as a group of networked
devices that cooperate to support their users. The
boundaries of a smart space are typically defined on the
basis of a geographic location, e.g. a room or a building.
However, such narrow definitions are not flexible enough
to support the application prototypes in the PECES
project. Obviously, these smart spaces cannot be defined on
the basis of a single location. For example in
applications based upon a car, the whole car, i.e. the
smart space itself, is mobile.

In order to define the scope of the cooperation
between devices, a set of basic groups is defined that
are residing in the same local network. This is a result of the
fact that the formation process of basic groups is limited
to a local network. Yet, for typical smart spaces
connectivity is guaranteed.

To support smart space formation, the PECES
middleware introduces three additional components
which are coordinator, member and gateway. These
components can be easily motivated by looking at the
anatomy of the smart spaces that are identified in the
PECES Use-Case Specification [4]:

- Coordinator: A smart space consists of at least one
  coordinator device. This device is responsible for
  identifying members of the smart space based on
  role specification.

- Member: In addition to coordinator device, a
  smart space may contain additional devices that
  are dynamically entering or leaving the local
  network. Depending on the context, a member
device might either be integrated or not. Currently,
a member device can only be integrated at most
into one smart space at a time.

- Gateway: Some devices that are part of a smart
  space may also be able to communicate with other
devices through an Internet connection. Examples
  for such devices are smart phones or residential
  gateways as well as laptops that are equipped with
  a UMTS modem. In these scenarios, the PECES
  middleware gateway functionality provides
  connectivity for other devices in the smart space.

4. Registry Interface Concept

The PECES middleware provides a collaboration
mechanism that enables communication between devices
in and across different smart spaces in a context
dependent manner. This requires a mechanism that allows
devices to discover and access information about each
other. The base middleware provides an internal service
and device registries to maintain the access information
of the locally available services. Since the services are
accessed not only from inside but also outside the smart
space, the PECES middleware provides a distributed
registry mechanism that can further be extended for
remote group formation. The distributed registry can
facilitate the information distribution of the assigned roles
in the smart spaces coupled with necessary context
information for forming an overarching environment. The
PECES cooperation layer can thus use the distributed
registry to lookup for devices based on the role and
retrieve necessary plug-in information to access the
devices. Hence, the distributed registry enables
cooperation between heterogeneous devices by
identifying the relevant devices that may provide services
or can be used to form an overarching smart space on top
of existing smart spaces.

The spontaneous appearance and disappearance of the
devices in a typical smart space naturally requires a
registry infrastructure where information about the
services and roles are easily but securely accessible. The
accessibility and security trade-off impose a natural
scoping on the service availability. From a device
perspective, the required services for an application may
reside on the same device, or may be available on a
remote device that may or may not be the part of the
same smart space. This clearly outlines three different
scopes for available services namely; “Device”, “Space”
and “Internet”. More detailed information about the
PECES registry concepts can be found in [8]

5. Security Concept

The PECES consortium extends the PECES
middleware to derive a secure middleware. For this,
PECES consortium introduced a basic trust model that is
used as basis for the concepts and mechanisms of the
middleware. These mechanisms enable the secure
interaction of devices. To enable this, they span the
management of cryptographic keys, the authentication of information – specifically context information and role assignments, the secure data and service centric communication as well as role based access control. Although they do not introduce additional interaction features, together they span the whole set of security related requirements that have been identified in the PECES Requirements Specification [5] and thus, they are sufficient to be applicable to a broad range of scenarios.

The PECES security mechanisms are modular and they introduce a certain degree of configurability that can be leveraged by application developers for optimization purposes. This enables them to define application specific trade-offs between security and application performance. In order to simplify the configuration of these mechanisms, the PECES consortium provides set of tools (which will be discussed in Section IV) that simplify basic security related tasks such as the distribution of keys and certificates during application development.

III. DEVELOPMENT ENVIRONMENTS

The PECES project provides a set of development tools that can be used by the application developers to develop, test, and analyse their applications. The Development tools also provide an environment to simulate/emulate applications. These types of development tools are economical because developers can carry out experiments without the actual hardware and it is a feasible way to test scalability of any proposed applications. For wired/wireless networked based application and protocol development, there are many simulators have been proposed.

Pervasive computing application developers need development tools for rapid development and evaluation of novel smart space systems application. This is especially true for dealing with heterogeneous device environments with context based smart space formation as proposed in the PECES prototype applications [4], [5]. To the best of our knowledge, not many frameworks are available for effective simulation, emulation and testing of smart space system applications. However, there are some general purpose development tools that are available to test/simulate pervasive computing based applications.

A Middleware based application framework for Active Space applications is proposed in [12]. The Active Space consists of the Gaia middleware OS [13] managing a distributed system composed of plasma displays, a video wall, audio system, touch screens, IR beacons, badge detectors, wireless and wired networks connecting several Windows 2000 and PDAs. The framework focuses on providing an application framework that leverages the functionality provided by the Gaia middleware OS to assist developers in the construction of Active Space application. The application framework defines an application model that accommodates the requirements of Active Spaces including dynamically changing the cardinality, location, input, output and processing devices used by an application. Then the application framework provides a mapping mechanism to define applications requirements and automatically mapping them to the resources present in a particular Active Space. Finally, the framework implements a flexible policy driven application management interface that allows customising applications to the dynamic behaviour of Active Spaces.

The Distributed Trust Toolkit (DTT) [11] proposed a framework for implementing and evaluating trust mechanisms in pervasive computing systems and introduced two new abstractions: trust groups and trust blocks. Trust groups allow associated application devices to share recorded trust data and trust computations. Trust blocks makes policy decisions based on data gathered by the computation component which implements network based trust protocols and allows the DTT to interoperate with legacy trust systems. The Distributed Trust Toolkit facilitates the extension and adaptation of trust mechanisms by abstracting trust mechanisms into interchangeable components. Furthermore, the DTT provides a set of tools and interfaces to ease implementation of trust mechanisms and facilitates their execution on a variety of platforms and networks.

UbiWise, a simulator for ubiquitous computing system was proposed in [15]. UbiWise concentrates on computation and communication devices situated within their physical environments. Multiple users can attach to the same server to create interactive ubiquitous computing scenarios. The devices are specified through a combination of a device-description file in XML and Java. UbiREAL simulator [16] was proposed for realistic smart space systematic testing. UbiREAL facilitates reliable and inexpensive development of ubiquitous applications where application software controls a lot of information appliances based on the state of external environment, user’s contexts information. The simulator realistically reproduces behaviour of application software on virtual devices in a virtual 3D space. Interestingly, it provides mechanisms to simulate virtual devices with real devices.

The tools discussed here provide limited support for application developers, as they have been designed for different goals and concepts. Although these tools are available to support pervasive computing environment application development, they only offer little methodological support for role assignment, context awareness and security, which are the core features of the PECES middleware. For example, Protégé [22] may provide support for context ontology instantiation for the devices but it may be difficult to integrate with other tools the way application developers wanted for smart space application development. The following sections present set of tools that have been developed for building novel smart space applications using the PECES middleware.

IV. PECES DEVELOPMENT TOOLS

PECES development tools focus on configuring devices, modelling smart spaces and context dynamics and testing the novel concepts provided by the PECES middleware. The tools provide support for application developers to build PECES middleware application and
simulate and analyse the smart space behaviours with respect to the context changes and network changes. Instead of running PECES application on real devices, application developers are able to test the features of the PECES middleware in a development PC for any specific application. This provides the opportunity for the application developers to test and analyse their application in a controlled and repeatable environment which enable them to optimise certain parameters which may be necessary for the best performance of any smart space applications.

The PECES development tools provide a set of tools which are integrated into the Eclipse development environment (as Eclipse Plugins). This way, the usual development assistance provided by the Eclipse IDE can also be used for PECES focused development support.

The following sections explain how to use the PECES development tools to build middleware applications. The tools are implemented as an Eclipse plugins and use a wizard and multipage editor approach. The development tools provide several tools for different purposes during application development. Earlier version of this tool was presented in [23]. The following sections explain the tools (listed in Figure 2) for configuration of the application such as Peces Project, Peces Device Definition, Peces Ontology Instantiation, Peces Security Configuration, Peces Service Definition, Peces Role Specification Definition, Hierarchical Role Specification Definition. For modelling purposes, Peces Event Editor and Peces Event Diagram tools are discussed. Finally for testing purposes, Peces Testing tool is explained. The following subsections are arranged according to the development sequence that the developers would typically follow during the smart space application development.

A. Peces Project Tool

This is the first part of the application development process. Using this tool, a project can be generated in the Eclipse workspace with three different folders to keep different configuration, modelling and testing related files which will be generated by other tools. For example, a PrototypeDemo project was created by the Peces Project tool with ConfigurationTool, ModellingTool and TestingTool folders (as shown in Figure 4).

B. Peces Device Definition Tool

The Peces Device Definition tool can be used to define communication plugins such as IP, Bluetooth, ZigBee (e.g. MxIPBroadcastTransceiver, MxIPMulticastTransceiver, MxSpotTransceiver), and device functionalities (Coordinator, Gateway, Coordinator&Gateway, Member) and also device name. All smart space applications should define one device with coordinator functionality which is responsible of the coorination of the smart space. The devices can be placed using drag and drop method. Different colours will be shown according to the selected device functionality (e.g., Coordinator is red).

Figure 3: Screenshot of the Peces Device Definition Tool

After placing the selected devices in the workspace, device IDs are automatically generated according to the order of the placement (e.g. first device placed in the workspace will be give to 1, next device will be give 2 and so on). This device ID will not be used by the middleware as device middleware address will be uniquely generated by the middleware based on the role specification mechanism but these IDs will be used to visualise the smart spaces based on the test log data. Application developers may change the device name and device communication features as well as device functionalities such as coordinator and gateway and member.

In this example application where five devices are defined namely GUIDESYSTEM, LOCATIONSYSTEM, VISITOR_HTC, TAXISYSTEM and TAXI. The GUIDESYSTEM and the TAXISYSTEM devices are defined as the coordinators of the network (shown as red in Figure 3), the LOCATIONSYSTEM and TAXI are defined as gateway (shown as green in Figure 3) and VISITOR_HTC is defined as member devices (shown as blue in Figure 3).

By completing the device definition process, five Java projects with the device name (GUIDESYSTEM,
LOCATIONSYSTEM, VISITOR_HTC, TAXISYSTEM and TAXI) are generated for each device by the tool (as shown in Figure 4). These Java projects contain PECES middleware libraries (peces-2.0.jar) and necessary Java files under the src folder. These Java projects are configured with the additional PECES Nature which automatically generates Java files from context definition (*.peces.ctx) file and SPARQL queries (*.peces.query) file.

C. Peces Ontology Instantiation Tool

The PECES context ontologies are composed by the SmartSpace, Measurement, Device profile, User Profile and Event ontologies and these ontologies are available at the PECES project website to download [17]. The document [6] clearly explains the dependencies among them, as well as the external ontologies which provide a basis for the PECES concepts and properties. The core ontology for representing contextual information of a smart space is the SmartSpace ontology. The basic concepts to model the contextual information of a smart space are Device, Context, Location and Service.

The Device profile ontology provides vocabularies to model specification of devices inside smart spaces. There are three categories of devices defined in the PECES prototype application which are PECSEmbeddedDevice, Accessory and SensorDevice. PECSEmbeddedDevice represents those embedded devices that deploy the PECES middleware. There are three categories of embedded devices according to their role inside a smart space, namely gateway, coordinator and member. In order to specify which kind of accessories an embedded device has, the property hasAccessory can be used to link a PECSEmbeddedDevice instance to an Accessory instance. Accessory instances are Keyboard, Touch Screen, Speaker, Screen and Microphone. In addition to this, SensorDevice has two sub-concepts: Detector and MeasuringSensor. A MeasuringSensor instance represents a sensor which can measure a measurement such as light, noise, temperature, etc.

Peces Ontology Instantiation tool enables application developer to instantiate the devices. This tool supports all PECES ontologies (e.g., http://www.ict-peces.eu/ont/device.owl) as well as other custom ontologies (e.g., http://www.daml.org/services/owl-s/1.1/Service.owl) which application developers may wish to use for instantiation of the devices. The Ontology Instantiation tool automatically loads the participating device name and its functionality information from the project.xml (Figure 4) file which is generated by the Peces Device Definition tool. The Peces Ontology Instantiation tool provides GUI where application developers can add instances and link properties. When the instantiation process is completed, the tool creates a RDF file (project.owl) in the PrototypeDemo project ConfigurationTool folder and also creates *.peces.ctx files which contains the device context information for each devices. Those device *.peces.ctx files are placed in the appropriate java project and once the is placed in the Java project, the PECES Nature will automatically create necessary Java files for the middleware from the *.peces.ctx and *.peces.query files. The *.peces.ctx files are used to provide local context information of the devices. The *.peces.query files are used to provide context queries which enable the smart space coordinator to defined roles to the smart spaces.

D. Peces Security Configuration Tool

The PECES middleware uses the OpenSSL library to create necessary certificates and keys. As a result, the Peces Security Configuration tool integrates the OpenSSL toolkit to enable application developers to generate keys and certificates for smart space application. The tool provides an interface to gather necessary information for root certificate, intermediate certificate (trust chain) and client certificate. The necessary information gathered from the Java interface is passed to

Figure 5: Screenshot of the Peces Ontology Instantiation Tool
the OpenSSL command line interface with the use of script files.

Developers should first generate a root certificate (as shown in Figure 6) and then are able to generate necessary trust chain and client certificates (as in shown Figure 7). Once necessary certificate chains are created, they appear as trees in the Certificates area. To generate a Client certificate, first, developers must select the appropriate trust chain in the tree, and then click on the “Client. Cert” button to generate client certificate. The tool also provides a mechanism for a device to deploy certificates to other devices which are to be trusted.

The “Device” option shows that which device will be implementing the selected service. The list of possible devices is shown to the developers, based on the definition in the Peces Ontology Instantiation tool. The Peces Service Definition tool automatically infers which the possible candidate is, but the developers can always change this default configuration.

The “Scope” option determines at which scope the service will be published. According to the PECES middleware Registry Interface Specification [8], the possible scopes are “Device” (available only to clients on the same device), “Space” (available to devices within the same smart space) and “Internet” (available to all smart spaces). The Peces Service Definition tool permits the developer to define the interface that the service will offer to its clients (i.e. the functions that will be available to them). These definitions follow a format that is similar to any Java function.

After completing this process, all necessary code is generated where needed. In particular, a Java file with the name of the service will be created in the project corresponding to the selected device. That file will contain an empty implementation of the service, which the developer will have to complete with the actual implementation.

E. Peces Service Definition Tool

The Peces Service Definition tool provides a simple interface to the developers that allows the automatic generation of necessary Java code needed to instantiate and make use of a PECES middleware based service.

The services defined (LocationService, TaxiService) in the Peces Ontology Instantiation tool is automatically loaded in the Peces Service Definition tool wizard page. The developer should select one service from the list of services to make use of the Peces Service Definition tool to further defined services. Once this process is completed, the Peces Service Definition tool main window will show several options to configure the service such as Device, Scope and Implemented Functions.

F. Peces Role Specification Definition Tool

The Peces Role Specification tool provides an interface where developers can define the different rules that the application will use to dynamically form groups of collaborative devices.

In the PECES middleware application, rules are written essentially as constrained queries over the context properties of the devices. For that reason, the Peces Role Specification Definition tool loads the results of the Peces Ontology Instantiation tool, showing on a tree-shaped diagram all the devices that have been defined in the project, and their properties (upon which the rules will be defined).

Using the tool, a new Role Specification can be defined. Application developers should select a device for Role Specification. The selected device functionality should be a coordinator. For these reason, a combo box with all coordinators defined for the project is presented, where developers can choose the proper one. There are three Registry Interface Specification, “Device” (available only to clients on the same device), “Space” (available to devices within the same smart space) and
“Internet” (available to all smart spaces). Developers should select one of the three available scopes depending on their application. Next, application developers should append Rulesets to the Role Specification. A Ruleset is a constrained query over the context properties of a device. A device fulfills a Ruleset when ALL the conditions defined there are fulfilled (AND conditions). On the other hand, a device fulfills a Role Specification when at least one of its Rulesets is fulfilled (OR conditions). Therefore, by combining several Rulesets in a single Role Specifications, reasonably complex conditions can be applied to the group formation process.

When a Ruleset is selected, its definition can be altered using the two tree-shaped property diagrams and the right hand window editor. By double clicking a property in the devices tree, a constraint over that property is added to the Ruleset. For instance, a constraint “?device provides ?service” would mean that “any device providing any service” fulfills the Ruleset. The window showing the defined constraints allow to change the third part of the constraints (“?service” in the example) by any possible value actually defined (possible values are shown in a combobox). For instance, a constraint “?device provides LocationService” would mean that “any device providing LocationService” fulfills the Ruleset. If the third part of a constraint is left undefined (i.e. beginning with an interrogation mark), it will appear in the variables tree. By appending a property of that variable to the Ruleset, composed constraints can be designed.

During the whole definition process, the bottom-left window with the title “Preliminary members of this smartspace” shows a prediction of the devices that fulfill the Role Specification, according to the static and initial context properties of all the devices defined within the project. By completing role specification process, the Peces Role Specification definition tool automatically generates all necessary Java code in the required projects to define and instantiate using the middleware.

Firstly, the application developers must choose the coordinator device which will in charge of the hierarchical smart space. A combo box listing all coordinators defined in the project is shown to the developers. Then application developers can choose which smart spaces will compose the hierarchical smart space. The developers can easily compose the list of “selected smart spaces” by using the appropriate buttons. By saving the Peces Hierarchical Role Specification, necessary files will be automatically created for smart space initializations and instantiations.

**Figure 9: Screenshot of the Role Specification Definition Tool**

G. Peces Hierarchical Role Specification Tool

The Peces Hierarchical Role Specification tool (as shown in Figure 10) provides an easy method to create all the code necessary to instantiate this kind of “composed” smart spaces. The tool presents a list with all the smart spaces included in the project.

**Figure 10: Screenshot of the Hierarchical Role Specification Definition Tool**

H. Peces Event Editor Tool

The Peces Event Editor tool is used to define single event definition. Five different events types can be defined which are Delay Events, Device SwitchON Event, Device SwitchOFF Event, Context Event and Connection Event. The Event Editor is a multipage editor and in the second page (the Context Page), the context of the corresponding device can be changed if the event’s type is Device Context Change. The third page (Connection Editor Page) can be used for connection related changes. The developers can only switch to this third page if the event’s type is Connection Link Change. The developers can create several *.peces.event files necessary for their test.

Figure 11 shows a connection event which generated for the example devices defined in the Peces Device Definition tool.
When the developers have defined the needed events using the Peces Event Editor tool, the sequence of the events can be easily defined with the Event Diagram Editor. The Peces Event Diagram tool enables application developer to model smart spaces, dynamics connections and dynamic contexts for testing. The export of the events which can be done with the icon next to the “+” icon will generate an events.xml file with all the information needed by the Peces Testing tool. Figure 12 shows Peces Event Diagram tool with the events generated for the example application described here.

The Peces Testing tool enables application developers to test the modeled application defined by the Peces Event Editor tool and Peces Event Diagram tool. For this purpose the Peces Testing tool generates another new Java project (emulator project) to centrally control other device related events. In addition to the necessary plugins, this new Java project should install the EmulationTransceiver plugin. This emulator Java project (run as new JVM process) is used to control connections and context dynamics (add and remove connections, add and remove context) between devices which is defined by the events.xml.

The Testing tool provides support for executing and analysing the smart space applications (each device application is considered as a separate JVM) built by the previous tools. The Testing tool loads and parses necessary device related information from the project.xml file and modelling information such as context changes connection changes from the events.xml file and also relevant device java project path information to execute the configured devices. The Testing tool wizard generates additional emulator Java project with the same name as the Testing tool editor to control the dynamic events of application. Application developers are able to define the required time to test the application specified by the previous tools. Developers can also provide Internet Registry IP and port information if they want to test that application with the internet registry. Figure 13 shows the defined application device status before the test was executed. The “Execute” button enables developers to run the application. As seen in above, the status of each device is shown during test (three devices are “ON” and one device is “OFF” at a particular time). All middleware and application related information is logged to a single log file with the specific device and absolute time of the system. Using this absolute time, the relevant time between the devices (based on test start time) are calculated for further analysis. The figure below shows the defined application device status while the test is running.

The Peces Testing tool provides features to visualise the smart space network status based on the test log data events with relative time. The Peces Testing tool analyses important events occurred during the test from the test log data and displays as a “List of Events”. The “List of Events” contain event information such Device Switch ON, Device Switch OFF, Connection, Disconnection and Smart Space Establish, Smart Space Join, etc. The status of the system can be viewed by double clicking on the name of the specific event. For example, by double clicking on the fourth event (TaxiBooking Establish) in the list in Figure 14 shows the visualisation of the smart space system at time 3050 ms (after test started). Figure 15 displays the two different smart spaces (BoothNavigation and TaxiBooking) with its coordinator devices and it also displays the PECES Internet Registry availability but not connected with the smart spaces at this time event. The smart spaces are expected to form a hierarchical smart space defined by the Peces Role Specification tool.
Figure 15 shows that *BoothNavigation* smart space and *TaxiBooking* smart space are formed a hierarchical smart space using the PECES Internet Registry when it is necessary. This clearly visualise the PECES middleware enables communication between devices in and across different smart spaces in a context dependent and secure manner.

**V. EVALUATION**

The PECES development tools have been evaluated by 20 evaluators from Germany, Spain and the UK. The evaluators were all Java programmers with varying degrees of experience ranging from Undergraduate Students thought to Post-Doctoral researchers and programmers from industry. Evaluators were given a short tutorial on PECES middleware and the development tools before testing the tools. They were asked to develop a simple service which required using the configuration tools (except for the Hierarchical Role Specification tool).

The evaluators completed a questionnaire and their development was measured by an Eclipse Productivity Plugin.

There were two main purposes of this evaluation. The first was to establish that user found the PECES development tools useful application development. Secondly, the tool developers wanted to obtain some useful suggestions as to how to improve the tools in the future.

**A. Result Analysis**

Figure 16 shows how difficult it was found to develop PECES middleware application without the PECES Development Tools. The range is 1-5 where 1 indicated very difficult and 5 indicated very easy. Developing applications on PECES middleware is thought to be difficult without development tools.

The following figure indicates the responses to the question “what is the general impression you have for the PECES Development Tools”. Range 1 indicated very impressive and 5 indicated very unimpressive. Over half delegates (55%) reported that the development tools are either impressive or very impressive. Only 25% people from the survey do not agree.

Figure 18 compares the ease of use of the different tools used. Every tool that was used is shown in the charts. The Range of responses was 1-5 where 1 indicated very easy to use and 5 indicated very difficult to use. Some tools are not considered easy to use because they require extra background knowledge or concepts from the middleware. The Ontology Definition tool, Role Specification tool and Hierarchical Role Specification tool were in this category.
There are still some room for improvements for the current version of the development tools. In this version, each device application is created as java project. There is no automatic support available to deploy the java applications on actually devices. We plan to improve the development tools and provide features to support deployment on different mobile platforms (e.g. Android).

PECES middleware supports building applications with multiple numbers of smart spaces. Even though all current the PECES Development Tools provide support for building application with multiple smart spaces except the Peces Testing Tool’s Visualization Page only supports two smart spaces visualisation at the moment. We are currently looking at the different options to provide support for visualizing multiple smart spaces.

ACKNOWLEDGEMENTS

The work presented here is sponsored by EC under FP7 programme (FP7-224342-ICT-2007-2) and authors also would like to thank all the project partners for their contributions.

REFERENCES


VI. FUTURE WORK AND CONCLUSIONS

One of the main objectives of the PECES middleware is to provide a cooperation layer that enables seamless interaction and coordination among devices in and across smart spaces in a secure manner. This paper presented a set of tools which provide support for PECES middleware based application development. The tools provide support for device configuration, ontology instantiation, security configuration and role specification. The tools also enable dynamic modelling of the network connections and context changes. Finally, the tools provide support to test the smart space application performance and visualise the test results. The Evaluation of the PECES Development Tools suggests that tools are useful for smart space application development. Tools are presented here already integrated with the evaluator’s comments.

Figure 19 depicts the percentage of users who agree that the PECES development tools are useful. 95% delegates believe development tools really useful for developing PECES middleware based application. Only one person thought that the tool set is unsuitable for application development.

Finally, 55% of the evaluators believe a high level of training is needed whereas 40% think they just need a normal amount of training.

These results suggest that the PECES Development Tools are useful for experienced developers of PECES application rather than general users without any background knowledge of the middleware.

B. Productivity Plugin

The aim of the productivity plug-in connected to Eclipse is to continuously track the different tasks developers work on, the time they individually require for carrying out those tasks, and the file operations needed to be performed within a given development environment. During the monitoring process the plug-in saves which file the developer edited and saved, when they started an application server, and which project or sub-project the user was involved in, and in which perspective.

The plugin measures the user interaction and hence the difficulty of use of each tool. The Role Specification tool was found to be the hardest to use, requiring almost twice as many interactions as the other tools.

Figure 19: Are PECES Development Tools Useful

Is PECES Development Tools useful ?

- Tools useful
- Tools not useful

95%


---

**Ran Zhao** is a doctoral student in Computing Science at Newcastle University. Before he started his Ph.D. study, he completed his M.Sc. in Computing Science from Newcastle University in 2007. Before that, he received his B.Eng. in Software Engineering from South China University of Technology, China.

**Kirusnapillai Selvarajah** completed his Ph.D. in Automatic Control and Systems Engineering at Sheffield University in August 2006. He obtained his B.Sc (Eng) in Electrical Engineering with 1st class honours from the University of Moratuwa, Sri Lanka in 2001. His research interests are Wireless Sensor Networks, Pervasive Computing, Embedded Systems, Swarm Intelligence and Optimisation. He worked as a Research Associate on the EU funded PECES and EMMA projects and currently working as a Research Fellow in the EPSRC funded MATCH project at The University of Nottingham.

**Neil Speirs** obtained a 1st class Honours degree in Mathematics from Newcastle University in 1980 and a doctorate in Theoretical Physics from the University of Durham in 1985. Since 1987, he has been at the University of Newcastle upon Tyne where he is currently a Senior Lecturer in Computing Science. His main research interests are in fault-tolerance, reliability and mobile distributed systems. He is the Newcastle University Project Manager on the EU PECES Project.