9

Intelligent Distributed eLearning Architecture

S. Stoyanov[1], H. Zedan[2], E. Doychev[1], V. Valkanov[1],
I. Popchev[1], G. Cholakov[1] and M. Sandalski[1]
[1]*University of Plovdiv,*
[2]*de Montfort University - Leicester Country*
[1]*Bulgaria*
[2]*UK*

1. Introduction

One of the main characteristics of the eLearning systems today is the 'anytime-anywhere-anyhow' delivery of electronic content, personalized and customized for each individual user. To satisfy this requirement new types of context-aware and adaptive software architectures are needed, which are enabled to sense aspects of the environment and use this information to adapt their behavior in response to changing situation. In conformity with [Dey,2000], a context is any information that can be used to characterize the situation of an entity. An entity may be a person, a place, or an object that is considered relevant to the interaction between a user and an application, including the user and the application themselves.

Development of context-aware and adaptive architectures can be benefited from some ideas and approaches of pervasive computing. Pervasive computing is a new paradigm for next-generation distributed systems where computers disappear in the background of the users' everyday activities. In such a paradigm computation is performed on a multitude of small devices interconnected through a wireless network. Fundamental to pervasive computing is that any component (including user, hardware and software) can be mobile and that computations are context-aware. As a result, mobility and context-awareness are important features of any design framework for pervasive computing applications. Context-awareness requires applications to be able to sense aspects of the environment and use this information to adapt their behaviours in response to changing situations.

One of the main goals of the Distributed eLearning Centre (DeLC) project [Ganchev, 2005] is the development of such an architecture and corresponding software that could be used efficiently for on-line eLearning distance education. The approach adopted for the design and development of the system architecture is focused on the development of a service-oriented and agent-based intelligent system architecture providing wireless and fixed access to electronic services and electronic content. This chapter provides a general description of the architecture for two types of access - mobile and fixed.

Furthermore, we present the Calculus of Context-aware Ambients (CCA in short) for the modelling and verification of mobile systems that are context-aware. This process calculus is

built upon the calculus of mobile ambient and introduces new constructs to enable ambients and processes to be aware of the environment in which they are being executed. This results in a powerful calculus where both mobility and context-awareness are first-class citizens. We present the syntax and a formal semantics of the calculus. We also present a new theory of equivalence of processes which allows the identification of systems that have the same context aware behaviours. We prove that CCA encodes the Pi-calculus which is known to be a universal model of computation.

We have used our CCA to specify DeLC in its entirety, hence achieving its correctness. Such a dynamic system must enforce complex policies to cope with security, mobility and context-awareness. We show how these policies can be formalised and verified using CCA. In particular an important liveness property of the mLearning system is proved using the reduction semantics of CCA.

2. DeLC overview

Distributed eLearning Center (DeLC) is a reference architecture, supporting a reactive, proactive and personalized provision of education services and electronic content. The DeLC architecture is modeled as a network (Fig.1.), which consists of separate nodes, called eLearning Nodes (eLNs). Nodes model real units (laboratories, departments, faculties, colleges, and universities), which offer a complete or partial educational cycle. Each eLearning Node is an autonomous host of a set of electronic services. The configuration of the network edges is such as to enable the access, incorporation, use and integration of electronic services located on the different eLNs.

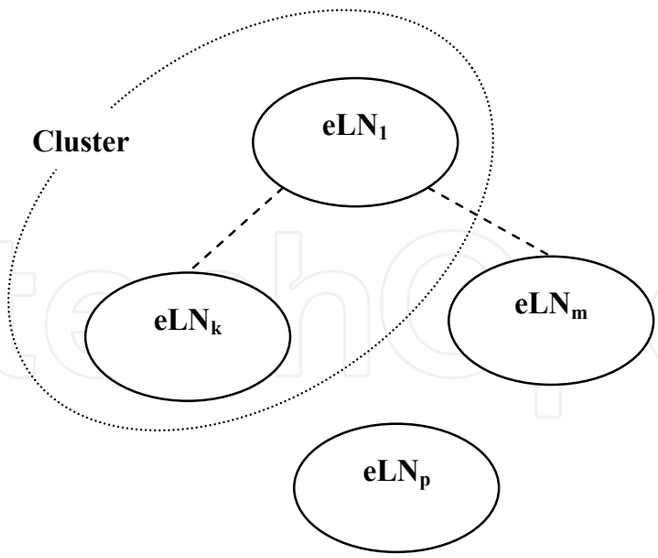


Fig. 1. DeLC Network Model

The eLearning Nodes can be isolated (eLN_p) or integrated in more complex virtual structures, called clusters. Remote eService activation and integration is possible only within a cluster. In the network model we can easily create new clusters, reorganize or remove

existing clusters (the reorganization is done on a virtual level, it does not affect the real organization). For example, the reorganization of an existing cluster can be made not by removing a node but by denying the access to the offered by it services. The reorganization does not disturb the function of other nodes (as nodes are autonomous self-sufficient educational units providing one or more integral educational services).

An important feature of the eLearning Nodes is the access to supported services and electronic content. In relation to the access there are two kinds of nodes:

- Mobile eLearning Node and
- Fixed eLearning Node.

For both nodes individual reference architectures are proposed within DeLC.

The current version of DeLC (Fig.2), two standardized architecture supporting fixed and mobile access to the eLearning services and teaching contend have been implemented. The fixed access architecture is adapted for the following domains implemented as particular nodes:

- Education portal supporting blended learning in the secondary school;
- Specialized node for electronic testing (DeLC Test Center);
- Specialized node for education in software engineering (eLSE);
- Specialized nod for examination of creative thinking and handling of students (CA). The node adapts the Creativity Assistant environment [Zedan,2008];

Intelligent agents that support the eLearning services provided by the DeLC portal (AV). The Agent Village will be presented in this chapter in more detail.

3. Mobile eLearning node

A distinguishable feature of contemporary mobile eLearning (mLearning) systems is the anywhere-anytime-anyhow aspect of delivery of electronic content, which is personalised and customised to suit a particular mobile user [Barker,2000], [Maurer,2001]. In addition, mobile service content is expected to be delivered to users always in the best possible way through the most appropriate connection type according to the always best connected and best served communication paradigm [O'Droma,2007], [Passas,2006]. In the light of these trends, the goal is to develop an intelligent mobile eLearning node which uses an InfoStation-based communication environment with distributed control [Frenkiel,1996], [Ganchev,2007]. The InfoStation paradigm is an extension of the wireless Internet, where mobile clients interact directly with Web service providers (i.e. InfoStations). By their mobile devices the users request services from the nearest InfoStation utilizing Bluetooth or WiFi wireless communication.

3.1 InfoStation-based network architecture

The continuing evolution in the capabilities and resources available within modern mobile devices has precipitated an evolution in the realm of eLearning. The architecture presented here attempts to harness the communicative potential of these devices in order to present learners with a more pervasive learning experience, which can be dynamically altered and

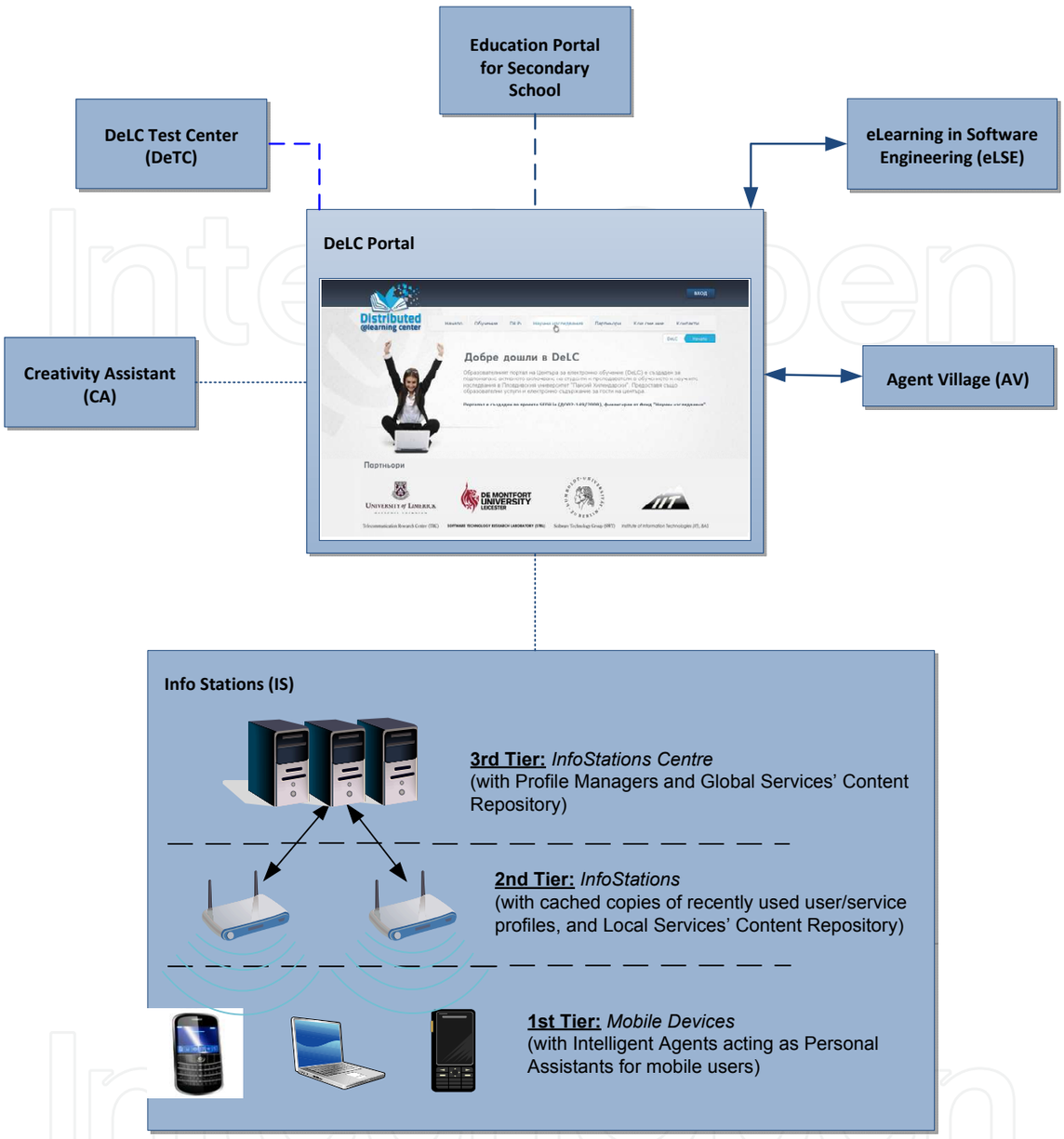


Fig. 2. Distributed eLearning Center

tailored to suit them. The following network architecture enables mobile users to access various mLearning services, via a set of intelligent wireless access points, or InfoStations, deployed in key points across the University Campus. The InfoStation-based network consists of three tiers as shown in Figure 3.

The first tier encompass the user mobile devices (cell phones, laptops, PDAs), equipped with intelligent agents acting as Personal Assistants to users. The Personal Assistant gathers information about the operating environment onboard the mobile device, as well as soliciting information about the user. Supplied with this information, the InfoStation can make better decisions on applicable services and content to deliver to the Personal Assistant.

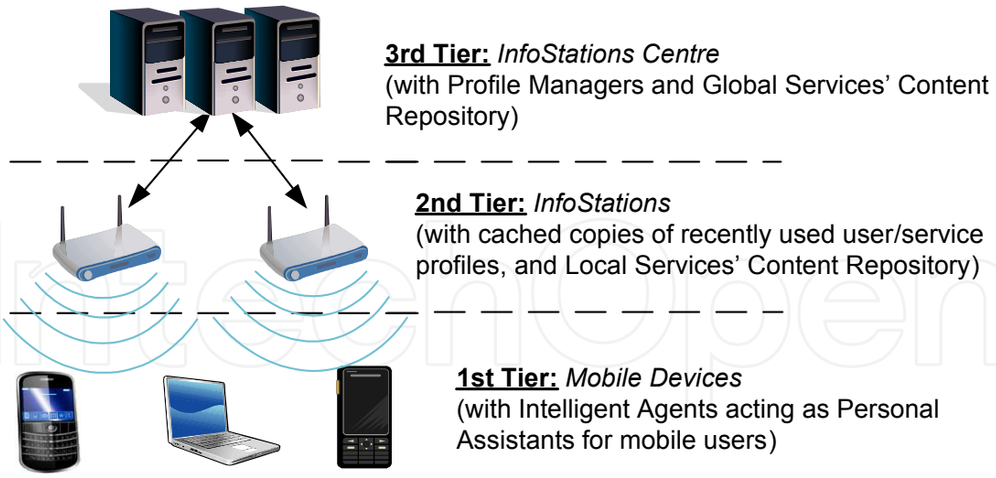


Fig. 3. The 3-tier InfoStation-based network architecture

The second tier consists of InfoStations, satisfying the users' requests for services through Bluetooth and/or WiFi wireless mobile connections. The InfoStations maintain connections with mobile devices, create and manage user sessions, provide interface to global services offered by the InfoStation Centre, and host local services. The implementation of these local services is an important aspect of this system. By implementing particular services within specific localised regions throughout the University campus, we can enrich the service users experience within these localities. A prime example of how this type of local service can enrich a learners experience, is the deployment of library-based services [Ganchev,2008a]. Within the library domain, library users experience can be greatly enhanced through the facilitation of services offering resource location capabilities or indeed account notifications. The division of global and local services allows for a reduction of the workload placed on the InfoStation Centre. In the original InfoStation architecture, the InfoStations operated only as mediators between the user mobile devices and a centre, on which a variety of electronic services are deployed and executed. The InfoStations within this architecture do not only occupy the role of mediators, they also act as the primary service providing nodes.

The third tier is the InfoStation Centre concerned with controlling the InfoStations, and overall updating and synchronisation of information across the system. The InfoStation Centre also acts as the host for global services.

3.2 Context-aware service provision

In order to ensure a context-aware service provision we propose that an application is built as an integration of two components [Stoyanov,2008]:

- A standardized **middleware**, which is able to detect the dynamic changes in the environment during the processing of user requests for services (*contex-awareness*) and correspondingly to ensure their efficient and non-problematic execution (*adaptability*);
- A set of **electronic services** realizing the functionality of the application area (education), which could be activated and controlled by the middleware.

As the middleware is concerned with the context-awareness and adaptability aspects, it is important to clarify these concepts. Within our development approach, Dey's definition [Dey,2000] was adopted, according to which "context is any information that can be used to characterize the situation at an entity". An entity could be a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. Context could be of different type, e.g. location, identity, activity, time.

Dey's definition is utilized here as a basis for further discussions. In order to elaborate on this definition a working one for the creation of the desired middleware architecture, we first solidify the definition as presented further in the chapter. We want clearly to differentiate context-awareness from the adaptability. Context-awareness is the middleware's ability to identify the changes in the environment/context as regards:

- Mobile device's location (*device mobility*) - in some cases this mobility leads to changing the serving InfoStation. This is especially important due to the inherent mobility within the system, as users move throughout the University campus. This information has a bearing on the local services deployed within a particular area i.e. within the University Library;
- User device (*user mobility*) - this mobility offers different options for the delivery of the service request's results back to the user. What is important here is to know the capabilities of the new device activated by the user, so as to adapt the service content accordingly;
- Communication type - depending on the current prevailing wireless network conditions/constraints, the user may avail of different communications possibilities (e.g. Bluetooth or WiFi);
- User preferences - service personalisation may be needed as to reflect the changes made by users in their preferences, e.g., the way the service content is visualised to them, etc.;
- Goal-driven sequencing of tasks engaged in by the user;
- Environmental context issues such as classmates and/or learner/educator interactions.

The goal of adaptability is to ensure trouble-free, transparent and adequate fulfilment of user requests for services by taking into account the various aspects of the context mentioned above. In other words, after identifying a particular change in the service environment, the middleware must be able to take compensating actions (counter-measures) such as handover of user service sessions from one InfoStation to another, re-formatting/transcoding of service content due to a change of mobile device (varying device capabilities), service personalisation, etc.

To ensure adequate support for user mobility and device mobility (the first two aspects of the context change), the following four main communications scenarios are identified for support in our middleware architecture [Ganchev,2008b]:

- '*No change*' - a mLearning service is provided within the range of the same InfoStation and without changing the user mobile device;
- '*Change of user mobile device*' - due to the inherent mobility, it is entirely possible that during an mLearning service session, the user may shift to another mobile device, e.g. with greater capabilities, in order to experience a much richer service environment and utilize a wider range of resources;

- '*Change of InfoStation*' - within the InfoStation paradigm, the connection between the InfoStations themselves and the user mobile devices is by definition geographically intermittent. With a number of InfoStations positioned around a University campus, the users may pass through a number of InfoStation serving areas during the service session. This transition between InfoStation areas must be completely transparent to the user, ensuring the user has continuous access to the service;
- '*Change of InfoStation and user mobile device*' - most complicated scenario whereby the user may change the device simultaneously with the change of the InfoStation.

To support the third aspect of the context change (different communication type), the development of an intelligent component (agent) working within the communication layer (c.f. Figure 4) is envisaged. This component operates with the capability to define and choose the optimal mode of communication, depending on the current prevailing access network conditions (e.g. congestion level, number of active users, average data rate available to each active user, etc.). The user identification and corresponding service personalisation is subject to a middleware adaptation for use in the particular application area. In the case of eLearning, the architecture is extended to support the three fundamental eLearning models - the educational domain model, the user/learner model, and the pedagogical model [Stoyanov,2005],[Ganchev,2008c].

3.3 Layered system architecture

The layered system architecture (Figure 4) is a distributed architecture, meaning that its functional entities are deployed across the different tiers/nodes, i.e. on mobile devices, InfoStations, and InfoStation Centre. In this architecture the role of the InfoStations is expanded, enabling them to act (besides the mediation role) as hosts for the local mLearning services (LmS) and for preparation, adaptation, and conclusive delivery of global mLearning services (GmS). This way the service provision is efficiently distributed across the whole architecture. Each of the system network nodes have a different structure depending on their functioning within the system. However, each node is built upon a Communication Layer whose main task is to initialize, control and maintain communications between different nodes. This layer is also concerned with choosing the most appropriate mode of communication between a mobile device and an InfoStation - whether that be Bluetooth or WiFi, or indeed as the platform evolves perhaps WiMAX in the future. The software architecture of the InfoStations and InfoStation Centre includes a Service Layer on the top. The main task of this layer is to prepare the execution of the users' service requests, to activate and receive the results of the execution of different services (local and global).

The InfoStations' middle layer is responsible for the execution of scenarios and control of user sessions. It is at this layer where the user service requests are mainly processed by taking into account all contex-aware aspects and applying corresponding adaptive actions. The middle layer of the InfoStation Centre ensures the needed synchronisation during particular scenarios (c.f. Section 8). In addition, different business supporting components, e.g. for user accounting, charging and billing, may operate here.

The software architecture of the user mobile devices contains two other layers:

- Personal Assistant - its task is to help the user in specifying the service requests sent to the system, accomplish the communication with the InfoStations' software, receive and visualise the service requests' results to the user, etc. Moreover the assistant can provide information needed for the personalisation of services (based on information stored in the user profile) and/or for the synchronisation of scenario execution;
- Graphical User Interface (GUI) - its task is to prepare and present the forms for setting up the service requests, and visualise the corresponding results received back from the system.

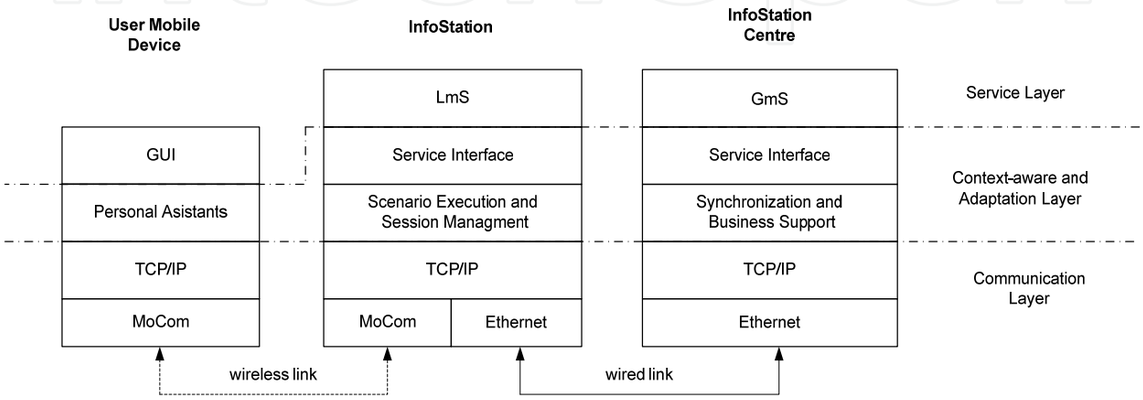


Fig. 4. The layered system architecture

3.4 Agent-oriented middleware architecture

The main implementation challenges within this system are related to the support of distributed control, as the system should be capable of detecting all relevant changes in the environment (context-awareness) and according to these changes, facilitate the service offerings in the most flexible and efficient manner (adaptability). The system architecture presented in the previous section is implemented as a set of cooperating intelligent agents. An agent oriented approach has been adopted in the development of this architecture in order to:

- Model adequately the real distributed infrastructure;
- Allow for realisation of distributed models of control;
- Ensure pro-active middleware behaviour which is quite beneficial in many situations;
- Use more efficiently the information resources spread over different InfoStations.

Moreover, the agent-oriented architecture can easily be extended with new agents (where required) that cooperate with the existing ones and communicate by means of a standardized protocol (in this case the FIPA -Agent Communication Language (ACL) [FIPA,2002]). Indeed the InfoStations and InfoStation Centre exist as networks of interoperating agents and services, with the agents fulfilling various essential roles necessary for system management. Within each of these platforms, agents take responsibility for selecting and establishing a client-server cross-platform connection,

conveyance of context information and the delivery of adapted and personalised service content. This multi-agent approach differs from the classic multi-tier architectures in which the relationships between the components at a particular tier are much stronger.

Conceptually we define different layers in the system architecture in order to present the functionality of the middleware that is being developed in a more systematic fashion. Implementation-wise, the middleware architecture is considered as a set of interacting intelligent agents. Communication between the user mobile devices and the serving InfoStations could be realized in two ways:

- An agent operating within the InfoStation discovers all new devices entering the range and subsequently initiates communication with them; or
- Personal Assistant agents on the user mobile devices are the active part in communication, and initiate the connection with the InfoStation.

In the current implementation of the prototype architecture, the former approach is used for Bluetooth communication, whereas the latter applies for WiFi communication.

Figure 4 highlights the main components necessary to ensure continuity to the service provision, i.e. support for the continuous provision of services and user sessions in the case of scenario change or resource deficiency. The agents which handle the connection and session establishment perform different actions, such as:

- Searching for and finding mobile devices within the range of an InfoStation;
- Creating a list of services required by mobile devices;
- Initiation of a wireless connection with mobile devices;
- Data transfer to- and from mobile devices.

Also illustrated within Figure 5 are the components which serve to facilitate a level of context sensitivity and personalisation to the presented services. A short description of the various agents (for Bluetooth communication) within the architecture is presented below.

The first step in the delivery of the services involves the Scanner agent, which continuously searches for mobile devices/Personal Assistant agents within the service area of the InfoStation. In addition, this agent retrieves a list of services required by users (registered on their mobile devices upon installation of the client part of the application), as well as the profile information, detailing the context (i.e. device capability and user preference information. The Scanner agent receives this information in the form of an XML file, which itself is extracted from the content of an ACL message. The contents of this XML file are then passed on via the Connection Advisor agent, to the Profile Processor agent, which parses the received profile and extracts meaningful information. This information can in turn be utilized to perform the requisite alterations to services and service content.

The information is also very important in relation to the tasks undertaken by the Scenario Manager agent. The role of this agent is to monitor and respond to changes in the operating environment, within which the services are operating (i.e. change of mobile device). In the event of a significant change of service environment, this agent gathers the new capability and preference information (CPI) via the Scanner agent. Then, in conjunction with the Query Manager agent and the Content Adaptation agent, facilitates the dynamic adaptation of the service content to meet the new service context.

The main duty of the Connection Adviser agent is to filter the list (received from the Scanner agent) of mobile devices as well as requested services. The filtration is carried out with respect to a given (usually heuristic) criterion. Information needed for the filtration is stored in a local database. The Connection Adviser agent sends the filtered list to the Connection Initiator agent, who takes on the task of initiating a connection with the Personal Assistant onboard the mobile device. This agent generates the so-called Connection Object, through which a communication with the mobile device is established via Bluetooth connection. Once this connection has been established, the Connection Initiator generates an agent to which it hands over the control of the connection, called a Connection agent.

From this point on, all communications between the InfoStation and the Personal Assistant are directed by the Connection agent. The internal architecture of the Connection agent contains three threads: an agent thread used for communication with the Query Manager agent, and a Send thread and Receive thread, which look after each direction of the wireless communication with the mobile device.

The Query Manager performs one of the most crucial tasks within the InfoStation architecture. It determines where information received from the mobile device is to be directed, e.g. directly to simple services, or via Interface agents to sophisticated services. It also transforms messages coming from the Connection agent into messages of the correct protocols to be understood by the relevant services, i.e. for simple services - UDDI or SOAP, or for increasingly sophisticated services by using more complicated, semantic-oriented protocols (e.g. OWL-S [OWL-S,2010]). The Query Manager agent also interacts with the Content Adaptation agent in order to facilitate the Personal Assistant with increasingly contextualised service content. This Content Adaptation agent, operating under the remit of the Query Manager agent, essentially performs the role of an adaptation engine, which takes in the profile information provided by the Profile Processor agent, and executes the requisite adaptation operations on the service content (e.g. file compression, image resizing etc.)

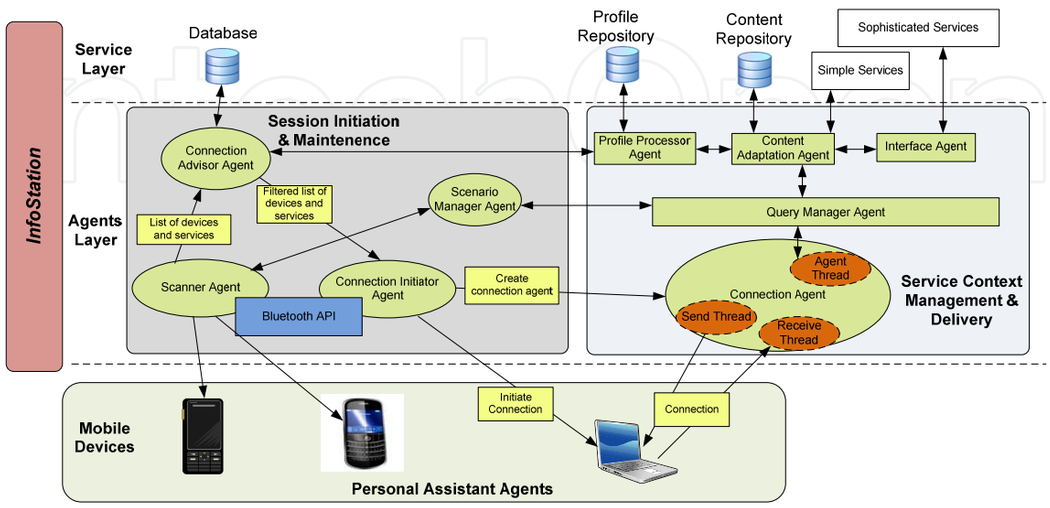


Fig. 5. The Agent-Oriented Middleware Architecture

The Query Manager agent receives user service requests via the Connection agent, and may communicate with various services. Once it has passed the request on to the services, all service content is passed back to the Query Manager via the Content Adaptation agent. The Profile Processor agent parses and validates received profiles (XML files) and creates a Document Object Model (DOM) tree [W3C,2010]. Using this DOM tree the XML information may be operated on, to discern the information most pertinent to the adaptation of service content. The Content Adaptation agent receives requests-responses from the services, queries the Profile Processor agent regarding the required context, and then either selects a pre-packaged service content package which closely meets the requirements of the mobile device, or applies a full transformation to the service content to meet the constraints of the operating environment of the device.

The tasks undertaken by the Content Adaptation agent, the Scenario Manager agent and the Profile Processor agent, enable the system to dynamically adapt to changing service environments, even during a particular service session. Once the connection to a particular service has been initialized and the service content adapted to the requisite format, the Connection agent facilitates the transfer of the information to the user mobile device.

4. Fixed eLearning node

The fixed nodes of the DeLC are implemented as education portals, which provides personalized educational services and teaching material. A standardized portal architecture is described in this section, which is used as generic framework for implementation of particular education portals for university and secondary school. The architecture has been extended by intelligent components (agents, called assistants) in order to enhance the flexibility, reactivity and pro-activeness of the portals.

4.1 Education portal architecture

The architecture of the educational portal is service-oriented and multi-layered, consisting of three logical layers (Figure 6): user interface, e-services and digital libraries.

The user interface supports the connection between the users and the portal. Through it the users can register in the system and create their own personalized educational environment. The user interface visualizes and provides access for the user to services, depending on their role, assigned during the registration.

Two kinds of e-services are located in the middle layer - system services and eLearning services. The system services, called 'engines', are transparent for the users and their basic purpose is to assist in the processing of the eLearning services. Using the information, contained in the meta-objects, they can effectively support the activation, execution and finalization of the eLearning services. In the current portal architecture the next engines are implemented:

- SCORM Engine;
- Exams Engine;
- Events and Remainders Engine;
- Integration Engine;
- User Profiling.

SCORM Engine is implemented in the portal architecture for delivering an interpreter of the electronic content, developed in accordance with the SCORM 2004 standard. The Test Engine assists in performing electronic testing using the portal. It processes basically the meta objects, which describe the questions and the patterns of the tests. The Event Engine supports a model for event management, enabling the users to see and create events and also be notified for them in advance. The events in the system reflect important moments for the users, such as a lecture, examination, test, national holiday, birthday, etc. One event is characterized by attributes, such as a name, start and end date and time, details, and information if it is a recurring one, as well as rules for its recurrence. The Event Engine supports yearly, monthly and weekly recurring. The User Profiling implements the user model of the portal. The profiles could be classified by roles, user groups, communities, and organizations. The standard user profile consists of three main groups of attributes:

- Standard attributes - necessary for user identification through username, password, e-mail, and others;
- Extended attributes - addresses, phone numbers, Internet pages, IM, social networks contacts, and others;
- DeLC custom attributes - other user identifications. Thus, for example, for users with role "student" these can be faculty number, subject, faculty, and course.

The portal gives an opportunity for extending the user profile with some additional attributes. The users' profiles contain the whole information needed for personalization of the provided by DeLC portal services, educational content and user interface. The profile is created automatically during the first user's log in, through a call to the university's database, filling in the standard and custom attributes. The integration with the university database and with ?nother external components is supported by the Integration Engine. Extended attributes are filled by the user. During each next user's log in in the portal the information in their profile is synchronized, as eventual updates in the university's database are automatically migrated in the user's profile, for example passage in the upper course or changing the subject.

Educational services serve all stages in one educational process. Supported by the portal, services are grouped in three categories:

- Services for training, organizing and planning of the educational process;
- Services for conduction and management of the education process - examples of these services are electronic lectures, electronic testing, online and offline consultations;
- Services for recording and documenting the educational process - these services support automated generation of the documents recording the educational process (examination protocols, student books, teachers' personal notebooks and archives).

The third layer contains electronic content in the form of repositories, known as digital libraries. In the current version are supported lecture courses digital library, questionary library, test templates library, course projects library and diploma theses library. The supported portal services work directly with the digital libraries. The digital libraries content can be navigated by help of a generalized catalog.

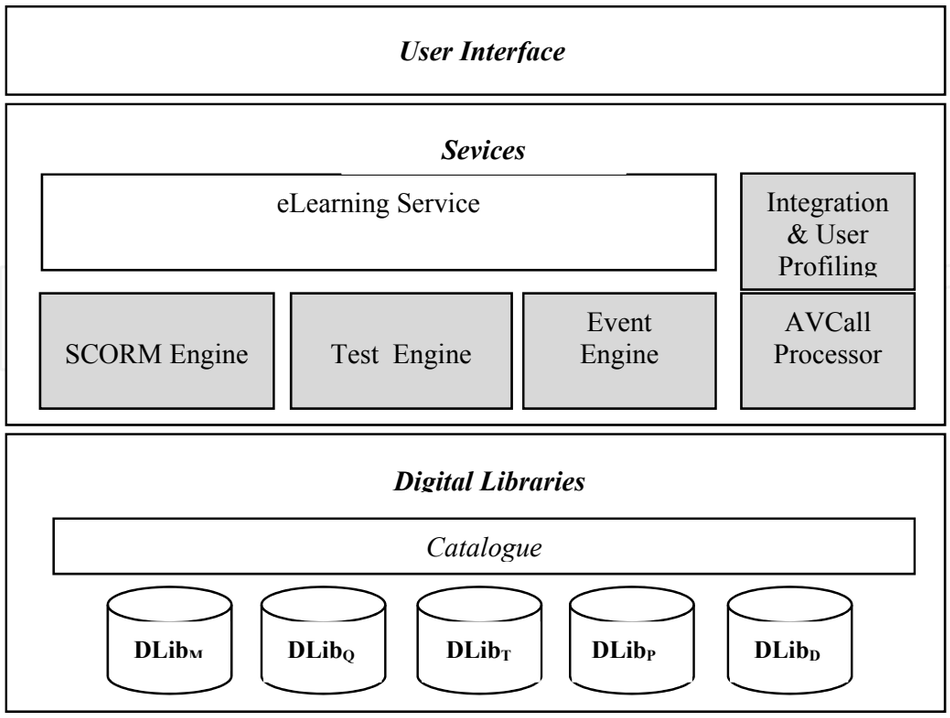


Fig. 6. Standarised Architecture of the portal

4.2 Education cluster

In order to provide more effective and personalized user support, we need to enhance the flexibility, reactivity and pro-activeness of the portal including intelligent components into the architecture. The pro-activity improves the usability and friendliness of the system to the users. Pro-activity means that the software can operate "on behalf" of the user" and "activate itself" when it "estimates" that its intervention is necessary. Two approaches are available:

- Direct integration of intelligent components in the currently existing architecture - in this way we extend the existing portal architecture;
- Building an education cluster.

The litter approach is preferable because it match with DeLC philosophy for building of more complex structures. Moreover, the former approach involves difficulties in the integration of two environments with different characteristics - portal frame and agent-oriented environment.

The education cluster consists of two nodes - the existing portal and a new node, called Agent Village (AV), where the "assistants" will "live in" (Figure 7). Three basic problems have to be solve in order to create the cluster:

- Architecture of the AV node;
- Interaction between the portal and AV;
- What kind of intelligent assistance for the portal services.

AV node is implemented as an agent-oriented server, by help of JADE environment [Bellifemine,2007].

The connection of the educational portal and the AV node is made through the middle layer of the portal architecture, where the electronic services are located. Depending on the direction of the asked assistance we distinguish reactive and proactive behavior of the architecture. In the reactive behavior the interaction between the two nodes is initiated by the portal. This is necessary in the cases when a user request is processed and a service needs an "expert" assistance. The service addresses the corresponding agent, located in the AV. The problem is that, in their nature, the services are passive and static software modules, intended mainly for the convenient realization and integration of some business functionality. Therefore they must "transfer" the responsibility for the activation and support of the connection to an active component of the architecture, as agents do. To do this, the service sends a concrete message to the agent's environment, which, on its behalf, identifies the change of the environment and reacts by interpreting the message. Depending on the identified need of assistance the agent activates the necessary actions. The reactive behavior of the architecture could be implemented using a:

- Synchronous model - this model is analogous to calling subroutines in programming languages. In this model the service sends a message to AV and waits for the result from the corresponding agent before continuing its execution.
- Asynchronous model - in the asynchronous model the interaction is accomplished through some kind of a mechanism for sending and receiving messages.

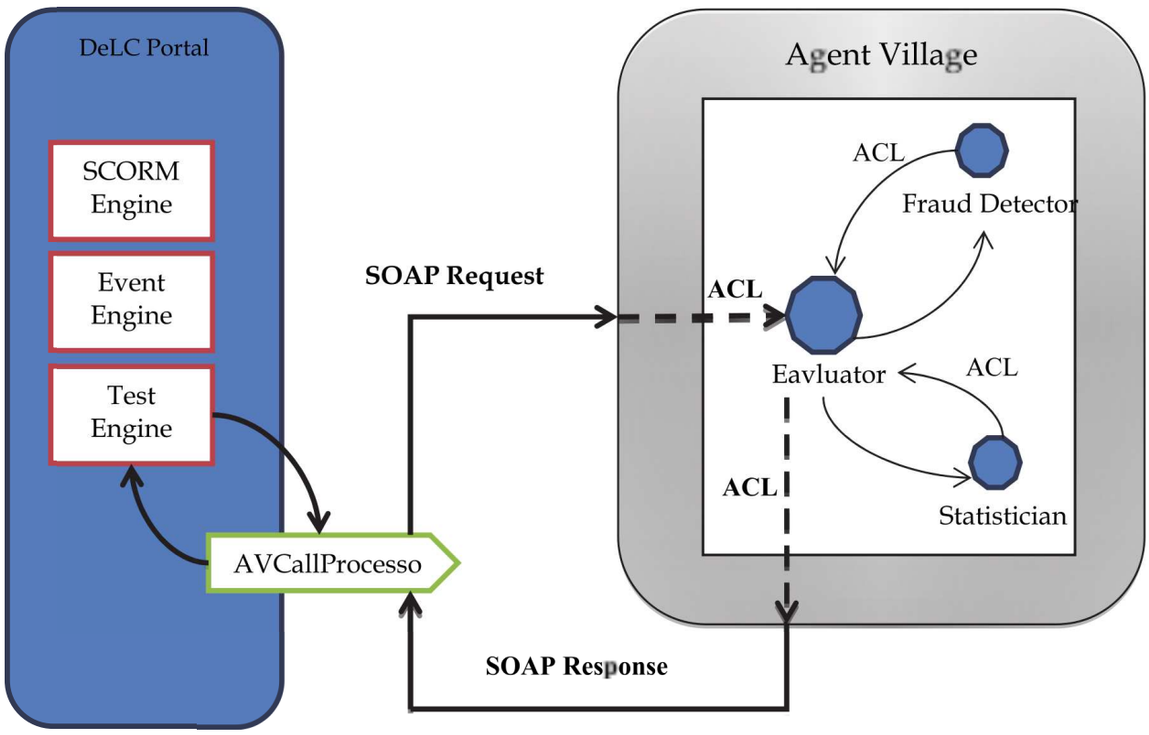


Fig. 7. Cluster architecture

In the proactive behavior (agents work "on behalf of the user"), an agent from the AV can determine that in its environment "something is happening", that would be interesting for the user, who is assisted by that agent. The agent activates and it can perform certain actions

to satisfy the preferences (wishes) of the user. The agent can inform the user of its actions through the educational portal.

The difficulties, associated with the management of the pro-activity of our architecture, result from the fact that the portal is designed for reaction of the user's requests. Therefore the pro-activity can be managed only asynchronously and for this purpose we provide development of a specialized service, which is to check a "mailbox'' periodically for incoming messages from AV.

According our architecture, the reactivity and the pro-activity are possible if the environment of the agents (Agent Village) remains not more passive. In order to be identified, the agents need a wrapper (the environment), which "masks'' it as a web service for the portal. In such a way the portal send the request to this service (masked environment), which in its turn transform the request into an ACL message, understandable for the agents. In a similar manner the active environment transform ACL messages into SOAP responses, which can be process from the portal services.

The next assistants are developing in the first version of the AV node:

- Evaluator Assistant (EA);
- FraudDetector;
- Statistician;
- Intelbos

The Evaluator Assistant (EA) provides expert assistance to the teacher in assessment of the electronic tests. In the Exam Engine a service is built for automated assessment of "choice like'' questions. In the standard version of the architecture questions of the "free text'' type are assessed by the teacher and the ratings are entered manually in the service to prepare the final assessment of the test. In the cluster the Exam Engine calls the assistant (an intelligent agent), which makes an "external'' assessment of the "free text'' type questions. In the surrounding environment of the EA, the received SOAP Request messages are transformed into ACL messages, understandable for the agent. Some of the basic parameters of the messages are:

- Text, which is an answer of a "free text'' type question.
- Parameters for the used estimation method.
- Maximum number of points for this answer.

The EA plans the processing of the request. In the current version of the assistant two methods are available for estimation:

- **Word Matching (WM)** method - counts "exact hits'' of the keywords in the answer. The minimum threshold of percentage match (i.e. a keyword to be considered as "guessed''), which is laid in the experiments, is between 70% and 80%. Intentionally, the method does not look for 100% match, in order to give a chance to words with some minor typos also to be recognized. To calculate the points, offered by this method, a coefficient is formed in the following way: the number of hits is divided by the number of keywords. The actual number of points for the answer is calculated as the maximum number of points is multiplied by this coefficient;

- **Optimistic Percentage (OP)** method - makes an optimistic estimation of the points for the answer. Its essence is to iterate over the keywords list and summarize their percentage matches. Thus, the calculated amount of rates for each keyword, divided by the maximum possible match (in %), gives the reduction coefficient. The actual number of points for the answer is calculated by multiplying the maximum number of points by the coefficient of the reduction. This method is more "tolerant" to allowing spelling mistakes in the answers, because low percentage matches are not ignored (unlike the first method) and are included in the formation of the final amount of points.

When the calculations finish, the EA generates an answer as an ACL message, which then is transformed by the environment into a SOAP Response message (a result from a web service call). In the answer there is a parameter, representing the calculated amount of points, extracted afterwards by the Exam Engine. A comparison of the scores, given by the two methods and by the teacher, are presented in Figure 8.

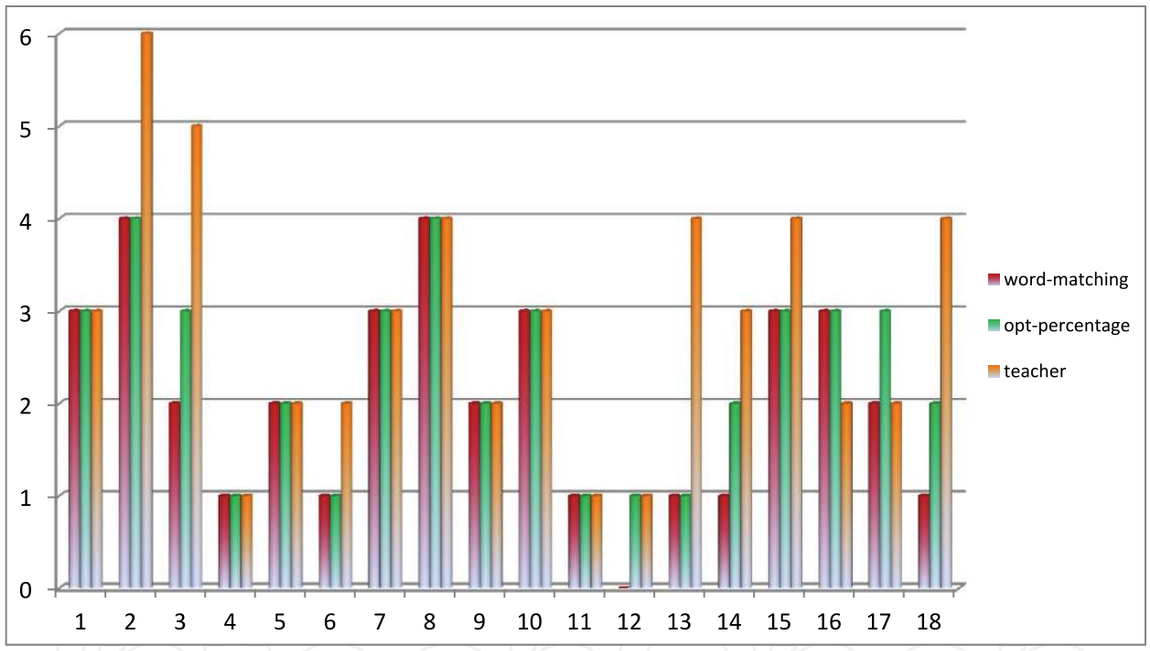


Fig. 8. Comparison of WM, OP and the teacher for 18 tests

The FraudDetector will try to recognize any attempts to cheat in the answer given by the student. Such attempts would be to guess the keywords or copy/paste results from Internet search engines. This assistant cooperates with the Evaluator agent and if its receptors detect a probability of a cheating attempt, it informs the Evaluator agent, which for its part informs the assessing teacher that this answer requires a special attention, because it is a suspicious one. The Statistician stores information about all processed answers with a full history of the details from all calculating methods used by the Evaluator agent. This assistant needs a feedback how many points are finally given by the teacher for each answer. Thus it accumulates a knowledge base for each teacher and is able to decide which of the methods best suits the assessment style of the current assessing teacher. Upon returning the results of the Evaluator assistant, information by this agent determines which results from each

method will be presented to the teacher as main result, and the results of the other methods will be presented as an alternative. Another feature of this agent will be also to provide actual statistics on the performance of each of the calculating methods, as the "weakest" of them goes out of service until new and better performing methods are added to the Evaluator agent. This monitoring of the methods' behavior becomes really significant when the so-called genetic algorithms are added, which we are still working on - as it is known, they can be "trained" and thus their effectiveness can change. In this process a knowledge base is developing for each specific subject, which supports the methods in their work.

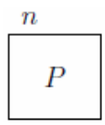
5. Calculus of context aware systems - CCA

Context-awareness requires applications to be able to adapt themselves to the environment in which they are being used such as user, location, nearby people and devices, and user's social situations. In this section we use small examples to illustrate the ability of CCA to model applications that are contextaware.

5.1 Syntax of processes and capabilities

This section introduces the syntax of the language of CCA. Like in the π-calculus [Milner,1999], [Sangiorgi,2001], the simplest entities of the calculus are *names*. These areused to name for example ambients, locations, resources and sensors data. We assume a countably-infnite set of names, elements of which are written in lower-case letters, e.g. n, x and y. We let $\tilde{y}$ denote a list of names and $|\tilde{y}|$ the arity of such a list. We sometimes use $\tilde{y}$ as a set of names where it is appropriate. We distinguish three main syntactic categories: processes P, capabilities M and context expressions κ.

The syntax of processes and capabilities is given in Table 1 where P, Q and R stand for processes, and M for capabilities. The first five process primitives (inactivity, parallel composition, name restriction, ambient and replication) are inherited from MA [Cardelli,2000]. The process 0 does nothing and terminates immediately. The process $P \mid Q$ denotes the process P and the process Q running in parallel. The process $(\nu n)\ P$ states that the scope of the name n is limited to the process P. The replication $!P$ denotes a process which can always create a new copy of P. Replication was first introduced by Milner in the π-calculus [Milner,1999]. The process $n[P]$ denotes an ambient named n whose behaviours are described by the process P. The pair of square brackets `[' and `]' outlines the boundary of that ambient. This is the textual representation of an ambient. The graphical representation of that ambient is:



The graphical representation highlights the nested structure of ambients.

CCA departs from MA and other processes calculi such as [Zimmer,2005], [Bucur,2008], [Bugliesi,2004] with the notion of *context-guarded capabilities*, whereby a capability is guarded by a context-expression which specifes the condition that must be met by the environment of the executing process. A process prefxed with a context-guarded capability is called a

context-guarded prefix and it has the form $\kappa?M.P$. Such a process waits until the environment satisfies the context expression κ, then performs the capability M and continues like the process P. The process learns about its context (i.e. its environment) by evaluating the guard. The use of context-guarded capabilities is one of the two main mechanisms for context acquisition in CCA (the second mechanism for context acquisition is the call to a process abstraction as discussed below). The syntax and the semantics of context expressions are given below. We let *M.P* denote the process **True**?*M.P*, where **True** is a context expression satisfied by all context.

P, Q, R ::=	**Process**
0	inactivity
P | Q	parallel composition
(υn) P	name restriction
n[P]	ambient
! P	repliaction
κ! M. P	context-guarder action
x ⊳ (ỹ). P	process abstraction
α ::=	**Locations**
↑	any parent
n ↑	parent n
↓	any child
n ↓	child n
::	any sibling
n ::	sibling n
ϵ	locally
M ::=	**Capabilities**
del n	delete n
in n	move in n
out	move out
α x⟨z̃⟩	process call
α (ỹ)	input
α ⟨ỹ⟩	output

Table 1. Syntax of CCA processes and capabilities

A process abstraction $x \rhd (\tilde{y}).P$ denotes the linking of the name x to the process P where $\tilde{y}$ is a list of *formal parameters*. This linking is local to the ambient where the process abstraction is defined. So a name x can be linked to a process P in one ambient and to a diferent process Q in another ambient. A call to a process abstraction named x is done by a capability of the form $\alpha\, x\langle\tilde{z}\rangle$ where α specifies the location where the process abstraction is defined and $\tilde{z}$ is the list of *actual parameters*. There must be as many actual parameters as there are formal parameters to the process abstraction being called. The location α can be $'\uparrow'$ for any parent, $'n\uparrow'$ for a specifc parent n, $'\downarrow'$ for any child, $'n\downarrow'$ for a specific child n, $'::'$ for any sibling, $'n::'$for a specific sibling n, or ϵ (empty string) for the calling ambient itself. A process call

$\alpha\, x\langle\tilde{z}\rangle$ behaves like the process linked to x at location α, in which each actual parameter in $\tilde{z}$ is substituted for each occurrence of the corresponding formal parameter. A process call can only take place if the corresponding process abstraction is available at the specified location.

In CCA, an ambient provides context by (re)defining process abstractions to account for its specific functionality. Ambients can interact with each other by making process calls. Because ambients are mobile, the same process call, e.g. $\uparrow x\langle\tilde{z}\rangle$, may lead to different behaviours depending on the location of the calling ambient. So process abstraction is used as a mechanism for context provision while process call is a mechanism for context acquisition.

Ambients exchange messages using the capability $\alpha\,\langle\tilde{z}\rangle$ to send a list of names $\tilde{z}$ to a location α, and the capability $\alpha\,(\tilde{y})$ to receive a list of names from a location α. Similarly to a process call, an ambient can send message to any parent, i.e. $\uparrow\langle\tilde{z}\rangle$; a specific parent n, i.e. $n\uparrow\langle\tilde{z}\rangle$; any child, i.e. $\downarrow\langle\tilde{z}\rangle$; a specific child n, i.e. $n\downarrow\langle\tilde{z}\rangle$; any sibling, i.e. $::\langle\tilde{z}\rangle$; a specific sibling n, i.e. $n::\langle\tilde{z}\rangle$; or itself, i.e. $\langle\tilde{z}\rangle$.

An *input prefix* is a process of the form $\alpha(\tilde{y}).P$, where $\tilde{y}$ is a list of variable symbols and P is a continuation process. It receives a list of names $\tilde{z}$ from the location α and continues like the process $P\{\tilde{y}\leftarrow\tilde{z}\}$, where $P\{\tilde{y}\leftarrow\tilde{z}\}$ is the process P in which each name in the list $\tilde{z}$ is substituted for each occurrence of the corresponding variable symbol in the list $\tilde{y}$.

The mobility capabilities in and out are defined as in MA [Cardelli,2000] with the exception that the capability out has no explicit parameter in CCA, the implicit parameter being the current parent (if any) of the ambient performing the action. An ambient that performs the capability in n moves into the sibling ambient n. The capability out moves the ambient that performs it out of that ambient parent. Obviously, a root ambient, i.e. an ambient with no parents, cannot perform the capability out. The capability del n deletes an ambient of the form $n[0]$ situated at the same level as that capability, i.e. the process del $n.P|n[0]$ reduces to P. The capability del acts as a garbage collector that deletes ambients which have completed their computations. It is a constrained version of the capability open used in MA to unleash the content of an ambient. As mentioned in [Bugliesi,2004], the open capability brings about serious security concerns in distributed applications, e.g. it might open an ambient that contains a malicious code. Unlike the capability open, the capability del is secure because it only opens ambients that are empty, so no risk of opening a virus or a malicious ambient.

5.2 Context model

In CCA the notion of ambient, inherited from MA, is the basic structure used to model entities of a context-aware system such as: a user, a location, a computing device, a software agent or a sensor. As described in Table 1, an ambient has a name, a boundary, a collection of local processes and can contain other ambients. Meanwhile, an ambient can move from one location to another by performing the mobility capabilities in and out. So the structure of a CCA process, at any time, is a hierarchy of nested ambients. This hierarchical structure changes as the process executes. In such a structure, the context of a sub-process is obtained by replacing in the structure that sub-process by a placeholder '$\odot$'. For example, suppose a system is modelled by the process $P|n[Q|m[R|S]]$. So, the context of the process R in that system is $P|n[Q|m[\odot|S]]$, and that of ambient m is $P|n[Q|\odot]$. Following are examples of

contexts in the smart phone system described in Sect. 5.3. The following context is the context of the smart phone carried by Bob when Bob is inside the conference room with Alice:

$$e_1 \triangleq conf\left[P \mid bob[\odot] \mid alice[Q]\right],$$

where P models the remaining part of the internal context of the conference room and Q the internal context of the ambient alice. We assume that there is only one ambient named *alice* in the conference room.

If Bob is inside the conference room while Alice is outside that room, the context of the smart phone carried by Bob can be described as follows:

$$e_2 \triangleq alice[Q] \mid conf\ \left[P \mid bob[\odot]\right].$$

Bob might carry with him another device, a PDA say, while inside the conference room. In this case the context of the smart phone can be modelled as:

$$e_3 \triangleq conf\left[P' \mid bob[\odot] \mid pda[R]\right],$$

where P' models the remaining part of the internal context of the conference room, pda is the name of the ambient modelling the PDA device and R specifies the functionality of the PDA.

Our context model is depicted by the grammar in Table 2, where the symbol E stands for context (environment), n ranges over names and P ranges over processes (as defined in Table 1). The context 0 is the empty context, also called the *nil* context. It contains no context information. The position of a process in that process' context is denoted by the symbol $\odot$. This is a special context called the *hole context*. The context $(\nu n)\ E$ means that the scope of the name n is limited to the context$E$. The context $n[E]$ means that the internal environment of the ambient n is described by the context E. The context $E|P$ says that the process P runs in parallel with the context E, and so E is part of process P's context.

Ground context. A ground context is a context containing no holes.

Note that a context contains zero or one hole; and that a ground context is a process. We do not allow multi-hole contexts because they are not suitable to our purpose.

E ::=	**Context**
0	nil
$\odot$	hole
$n[E]$	location
$(\nu n)\ E$	restriction

Table 2. Syntax contexts

Context evaluation. Let E_1 and E_2 be contexts. The evaluation of the context E_1 at the context E_2, denoted by$E_1(E_2)$, is the context obtained by replacing the hole in E_1 (if any) by E_2, viz

$$E_1(E_2) = \begin{cases} E_1 & \text{if } E_1 \text{ is a ground context} \\ E_1\{\odot \leftarrow E_2\} & \text{otherwise.} \end{cases},$$

where $E_1\{\odot \leftarrow E_2\}$ is the substitution of E_2 for $\odot$ in E_1.

The hole $\odot$ plays an important role in our context model. In fact a context E containing a single hole represents the environment of a process P in the process $E(P)$. A process modelling Bob using a smart phone in the conference room with Alice can be specified as:

$$e_1(phone[S]) \triangleq conf\left[P \mid bob[phone[S]] \mid alice[Q]\right],$$

where e_1 is the context specified in Example 5.2 and S is the specification of the smart phone.

A process modelling Bob using a PDA in the conference room can be specified as:

$$e_3(0) \triangleq conf\left[P' \mid bob[pda[R]]\right],$$

where e_3 is the context specified in Example 5.2. The syntax of CEs is given in Table 3 where κ ranges over CEs, n ranges over names and x is a variable symbol which also ranges over names.

κ ::=		**Context Expressions**
	True	true
	$n = m$	name match
	$\bullet$	hole
	$\neg\kappa$	negation
	$k_1 \mid k_2$	parallel composition
	$k_1 \wedge k_2$	conjunction
	$n[\kappa]$	location
	$\text{new}(n, k)$	relevation
	$\oplus\kappa$	spatial next modality
	$\diamondsuit\,\kappa$	somewhere modality
	$\exists x.\,\kappa$	existential quantification

Table 3. Syntax context expressions

5.3 A simple example

This example illustrates the use of process abstraction and process call as a mechanism for context provision and context acquisition, respectively. A process abstraction can be thought of as the declaration of a procedure in procedural programming languages and a process call as the invocation of a procedure.

A process abstraction links a name x to a process P using the following syntax:

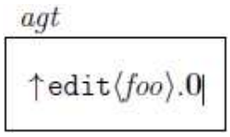
$$x \rhd (\tilde{y}).P$$

where $\tilde{y}$ is the list of formal parameters. A process call to this process abstraction has the following syntax:

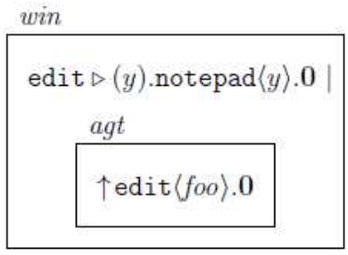
$$x\langle\tilde{z}\rangle$$

where $\tilde{z}$ is the list of the actual parameters. This process call behaves exactly like the process P where each actual parameter in $\tilde{z}$ is substituted for each occurrence of the corresponding formal parameter in $\tilde{y}$. In the smart phone example presented above, switchto is a process abstraction.

Suppose a software agent agt (here modelled as an ambient) is willing to edit a text file *foo*. This is done by calling a process abstraction named edit say, as follows:



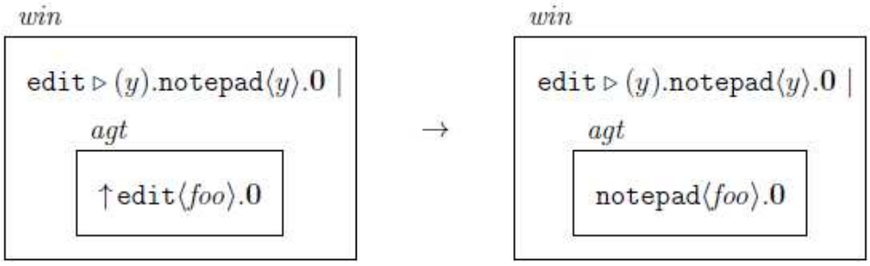
where the symbol ↑ indicates that the edit process called here is the one that is defined in the parent ambient of the calling ambient *agt*. Now suppose agent *agt* has migrated to a computing device win running Microsoft Windows operating system:



On this machine, the process abstraction edit is defined to launch the text editor notepad as follows:

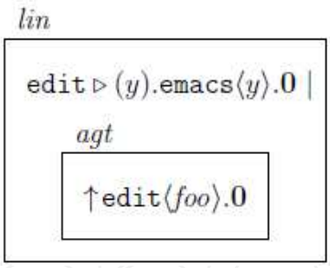
$$\text{edit} \triangleright (y).\text{notepad}\langle y\rangle.\,0.$$

So the request of the agent *agt* to edit the fille *foo* on this machine will open that file in notepad according to the following reduction:

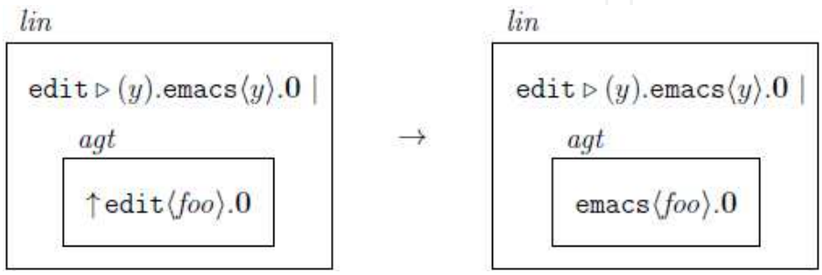


Note that the command notepad has replaced the command edit in the calling ambient *agt*.

Now assume the agent *agt* first moved to a computer *lin* running linux operating system:



On this computer, the command edit is configured to launch emacs. So in this context, the file *foo* will be opened in emacs as illustrated by the following reduction:



Our agent *agt* might have even moved to a site where the command edit is not *available* because no process abstraction of that name is defined. In this case the agent *agt* will not be able to edit the file *foo* at this site and might consider moving to a nearby computer to do so.

6. InfoStation-based mLearning system

As we have mentioned earlier that eLearning is becoming an authentic possible alternative educational approach as the technologies regarding that area are developing so fast, and there is a recognisable growth of a great variety of wide-band telecommunication delivery technologies. The infostation paradigm first proposed by Frenkiel et al. [Frenkiel,1996] and used in [Ganchev,2007] to devise an infostation-based mlearning system which allows mobile devices such as cellular phones, laptops and personal digital assistants (PDAs) to communicate to each other and to a number of services within a university campus. This mLearning system provides a number of services among which are: mLecture, mTutorial, mTest and communication services (private chat, intelligent message notification and phone calls). This section presents the architecture of the infostation-based mLearning system and describes the policies of the mLecture service.

6.1 mServices

This section introduces at glance each of the mServices provided by the infostation-based mLearning system.

- *AAA*: in order for any user to use any mService in the system, the user device should be registered. The AAA service (Authentication, Authorisation and Accounting) allows the users to register their devices with the system to gain the ability of using the mLearning services offered by the system.

- *mLecture*: this service allows the users to gain access to the lecture material through their mobile devices. The users can request a specific lecture, which is adapted according to the capabilities of the user devices and then delivered to their mobile devices.
- *mTest*: this service is crucial to the learning process. The mTest service allows the users to gain access to test materials that provide means of an evaluating process. A user can request, like the mLecture service, a specific test, which is also adapted to the capabilities of the user device then delivered to the user mobile device. The mTest service may only runs individually on a user device and unaccompanied with any other service whatsoever.
- *mTutorial* : this service allows the users to gain access to a self-assessment test. It is a combination between the mLecture and the mTest services. A user can request a self-assessment test in a similar way as requesting a mLecture. After the user submits their answers, he receives a feedback on his performance and the correct answers to the questions he got wrong.
- *Intelligent Message Notification (IMN in short)*: this service allows the users to communicate with each other by exchanging messages via their mobile devices.
- *VoIP*: this service allows the users to communicate with each other via phone calls throughout the infostation-based mLearning system.

6.2 Policies

The InforStationCentre (ISC) provides the User Authentication, Authorisation and Accounting (AAA) service which identifies each mobile user and provides him with a list of services the user is authorised to access. This service is regulated by the following policies:

- When a user is within the range of an IS, the intelligent agent (PA) of the user's device and the IS mutually discover each other. The PA sends a request to the IS for user Authorisation, Authentication and Accounting (AAA). This request also includes a description of the mobile device currently being used and any updates of user pro_le and user service profile.
- The IS forwards this AAA request to the ISC along with the profile updates. If the user is successfully authenticated and authorised to utilise the services by the AAA module within the ISC, a new account record is created for the user and a positive acknowledgement is sent back to the IS. Then the IS compiles a list of applicable services and sends this to the PA along with the acknowledgement. The PA displays the information regarding these services to the user who then makes a request for the service he wishes to use.

If the user chooses the mLecture service, then the following policies of the mLecture service apply:

- The PA forwards the mLecture service request to the InfoStation, which instantiates the service. If the IS is unable to satisfy fully the user service request it is forwarded to the ISC, which is better equipped to deal with it. In either case, the lecture is adapted and customised to suit the capabilities of the user devices and the user own preferences, and then delivered to their mobile devices.

- During the execution of the service, the user is free to move into a different infostation, to switch between devices or to do both.
- A user cannot use the mLecture and mTest services simultaneously. The mTest service should operate unaccompanied at all occasions.

This section presents the formalisation of the policies of the infostation-based context-aware mLearning system using . We first introduce some naming conventions (sect. 6.2.1) which are used in the specification of the system. Then we give the specification of two mServices which are AAA and the *mLecture* services (sect. 6.3).

6.2.1 Notations

The following naming conventions are used to differentiate between variables' names and constants. A variable name begins with a lowercase letter while a constant begins with a number or a uppercase letter. The list of the constant names that are used in the formalisation process is given in Table 4. And the list of variable names is given in Table 5.

Notations	Descriptions
ISC	the InfoStation Centre
IS_i	the i-th InfoStation
$Phone$	a phone
PDA	a PDA
PC	a PC
$SLIST$	list of mServices
ACK	an acknowledgement
NULL	empty message
DENIED	request denied

Table 4. Constants

6.3 A Model of the InfoStation-based mLearning system

The system consists mainly of one central ISC, multiple ISs and multiple user devices. Each component of the system is modelled as an ambient. That is, the ISC, each IS and each user device is modelled as an ambient. In particular, a device, *PC* say, being used by a user, *303* say, is modelled by an ambient named *PC303*. The ISC ambient runs in parallel with the IS ambients, and all the user devices within the range of an IS are child ambients of that IS ambient.

uid 301, 302, 303	a user's ID
dtype Phone, PDA, PC	a user's device type
aname Phone301, PC303	an ambient's name
lect Lect001	a lecture's ID
reply OK, DENIED, content	a reply to a request
content CONTENT	lecture's content
slist SLIST	list of services
ack ACK	acknowledgement

Table 5. Variabbles

This is textually represented by the following process:

$$\begin{aligned} ISC[P_{ISC}] \;\mid\; & IS1[PDA303[P_{PDA303}] \mid PC401[P_{PC401}] \\ & \mid P_{IS1}] \\ \mid\; & IS2[PDA301[P_{PDA301}] \\ & \mid Phone402[P_{Phone402}] \mid P_{IS2}] \\ \mid\; & IS3[Phone300[P_{Phone300}] \mid P_{IS3}] \\ & \mid IS4[Phone403[P_{Phone403}] \mid PC302[P_{PC302}] \\ & \mid P_{IS4}] \end{aligned} \tag{1}$$

where each P_x is a process modelling the behaviour of the corresponding ambient x.

Now we give the formal specification of the ISC and the ISs below.

InfoStation An abstract model of an infostation IS_i (for some integer i) has the following main components are the AAA request ambient $AAAreq_i$, the lecture ambient $Lectreq_i$ and the cache ambient $Cache_i$.

The InfoStation is a parent to the inside ambients which are siblings to each other. The specification of each of these ambients is as follows:

$\boldsymbol{AAAreq_i}$ This ambient is responsible for handling AAA requests sent by user devices willing to register with the InfoStation IS_i. The $AAAreq_i$ ambient receives a request from a device and, immediately, forwards it to the InfoStation, then receives a reply from the InfoStation and again, forwards it to the user's device. This behaviour is modelled by the following process:

$$P_{A_i} \triangleq ! :: (uid, dtype, aname).IS_i \uparrow \langle uid, dtype, aname\rangle.IS_i \uparrow (ack, aname, slist).aname :: \langle aname, slist\rangle.0$$

where *uid* is the user ID, *dtype* is the device type and *aname* is the name of the ambient sending the request.

The InfoStation accordingly receives a request from the $AAAreq_i$ ambient, forwards it to the InfoStation Centre, and after receiving the reply from the InfoStation Centre it forwards it to the $AAAreq_i$ ambient. This behaviour is modelled as:

$$\left(\begin{array}{l} !AAAreq_i\downarrow(uid, dtype, aname).ISC :: \langle AAAreq, uid, \\ dtype, aname, IS_i\rangle.\mathbf{0} \mid \\ !ISC :: (ack, aname, slist).(\mathtt{has}(aname))? \\ AAAreqi\downarrow\langle ack, aname, slist\rangle.\mathbf{0} \end{array}\right) \quad (2)$$

$\boldsymbol{Lectreq_i}$ This ambient handles all the mLecture service requests sent by the user devices. It receives a lecture request from a user device and forwards it to the infostation IS_i, i.e.

$$! :: (lectid, uid, dtype, aname).IS_i \uparrow \langle lectid, uid, dtype, aname\rangle.0 \quad (3)$$

Then it gets the reply from that infostation and forwards it to the user device which initiated the request, i.e.

$$! :: (lectid, reply, aname).aname :: \langle lectid, reply\rangle.0 \quad (4)$$

So the whole behaviour of the $Lectreq_i$ ambient is

$$P_{L_i} \triangleq Eq.(3) \mid Eq.(4) \quad (5)$$

We show how the InfoStation handles a request from the $Lectreq_i$ ambient after we have specified the $Cache_i$ ambient.

$\boldsymbol{Cache_i}$ This is the ambient where the InfoStation stores copies of requested lectures for future rapid access. It models a cache memory. A lecture is stored as an ambient (named after that lecture's id) which contains three persistent memory, each containing a version of the lecture suitable to a specific type of device (phone, PDA or PC). When an InfoStation receives a mLecture service request from a device, it checks for the requested material in its cache first rather than getting it from the InfoStation Centre directly. The process of checking the availability of a lecture inside the cache is done by sending a request to the $Cache_i$ ambient which then checks whether it has the ambient of the requested lecture or not. If the requested lecture is available the cache ambient retrieves it and sends it back to the InfoStation, otherwise, it replies immediately to the InfoStation that this lecture does not exist. The behaviour of the $Cache_i$ ambient is modelled by the following process:

$$P_{c_i} \;\widehat{=}\; !\uparrow(lectid, uid, dtype, aname).\left(\begin{array}{l}\texttt{has}(lectid)?lectid\downarrow\langle dtype, aname\rangle.\\ lectid\downarrow(reply, aname).\\ \uparrow\langle lectid, uid, dtype, reply, aname\rangle.\mathbf{0}\mid\\ \neg\texttt{has}(lectid)?\uparrow\langle lectid, uid, dtype,\\ NULL, aname\rangle.\uparrow(lectid, content, dtype).\\ lectid\downarrow\langle content, dtype\rangle.lectid\downarrow(ack).\uparrow\langle ack\rangle.\mathbf{0}\end{array}\right) \tag{6}$$

The behaviour of each lecture ambient (named after the lecture's id *lectid*) in the cache is modelled by the following process:

$$P_{lectid} \;\widehat{=}\; !\uparrow(dtype, aname).\left(\begin{array}{l}\texttt{has}(dtype)?dtype\downarrow\langle\rangle.\ dtype\downarrow(reply).\\ \uparrow\langle reply, aname\rangle.\ \mathbf{0}\mid\\ \neg\texttt{has}(dtype)?\uparrow\langle NULL, aname\rangle.\\ \uparrow(content, dtype).dtype\downarrow\langle content\rangle.dtype\downarrow().\\ \uparrow\langle ACK\rangle.\mathbf{0}\end{array}\right) \tag{7}$$

The InfoStation will act as follows. First, it receives a request from $Lectreq_i$, then it checks the availability of the lecture in its cache by sending a request to the $Cache_i$ ambient, i.e.

$$!\,Lectreq_i\downarrow(lectid, uid, dtype, aname).Cache_i\downarrow\langle lectid, uid, dtype, aname\rangle.0 \tag{8}$$

If the cache replies with the content of the lecture, it will send a request to the InfoStation Centre with a flag set to 1 (meaning that the requested lecture exists in its cache) asking whether the user is currently taking a mTest. If the user is taking a mTest, then the mLecture service request must be denied. If the cache did reply with NULL as lecture's content, then the infostation will send a request to the InfoStation Centre with the flag set to 0 (meaning that the lecture does not exist in its cache) asking for both the requested lecture and to check whether the user is taking a mTest. This behaviour of the IS is modelled as:

$$!Cache_i\downarrow(lectid, uid, dtype, creply, aname).\left(\begin{array}{l}\neg(creply = NULL)?\ ISC :: \langle lectid, uid, dtype,\\ aname, 1\rangle.\mathbf{C}\mid\\ (creply = NULL)?\ ISC :: \langle lectid, uid, dtype,\\ aname, 0\rangle.\mathbf{N}\end{array}\right) \tag{9}$$

where C and N are de_ned as follows:

$$\mathbf{C} \;\widehat{=}\; ISC :: (lectid, reply, aname).\left(\begin{array}{l}(reply = OK \wedge \texttt{has}(aname))?\\ Lectreq_i\downarrow\langle lectid, creply, aname\rangle.\mathbf{0}\mid\\ (reply = OK \wedge \neg\texttt{has}(aname))?\\ ISC :: \langle lectid, uid, dtype, aname, 0\rangle.\mathbf{0}\mid\\ (reply = DENIED \wedge \texttt{has}(aname))?\\ Lectreq_i\downarrow\langle lectid, reply, aname\rangle.\mathbf{0}\end{array}\right)$$

and

$$\mathbf{N} \;\widehat{=}\; ISC :: (lectid, aname, reply).\left(\begin{array}{l}(\neg(reply = DENIED) \wedge \mathtt{has}(aname))?\\ Cache_i{\downarrow}\langle lectid, reply, dtype\rangle.Lectreq_i{\downarrow}\langle lectid,\\ reply, aname\rangle.\ Cache_i{\downarrow}(ack).\mathbf{0}\ |\\ (\neg(reply = DENIED) \wedge \neg\mathtt{has}(aname))?\\ Cache_i{\downarrow}\langle lectid, reply, dtype\rangle.ISC :: \langle lectid, uid,\\ dtype, aname, 0\rangle.\ Cache_i{\downarrow}(ack).\mathbf{0}\ |\\ (reply = DENIED \wedge \mathtt{has}(aname))?\\ Lectreq_i{\downarrow}\langle lectid, reply, aname\rangle.\mathbf{0}\end{array}\right)$$

Thus, the whole behaviour of an infostation IS_i is modelled as

$$P_{IS_i} \triangleq Eq.(2) \mid Eq.(8) \mid Eq.(9) \tag{10}$$

6.3.1 InfoStation centre

A model of the ISC encompasses ambients modelling users' accounts and named after the users' IDs; an ambient named *Lectures* that contains all the lecture ambients, each named after the corresponding lecture ID Each lecture ambient contains three persistent memory cells named *Phone*, *PDA* and *PC*; each storing the lecture's version suitable for the corresponding type of device. The mTest service is not represented as we are only dealing with the mLecture service in this paper. These components of the ISC are formalised below.

Users' accounts An ambient modelling a user's account contains two ambients named *Loc* and *Utest*. Each of these two ambients models a persistent memory cell which stores, at any time, the current location of that user (for the former) or a Boolean indicating whether that user is taking a mTest or not (for the latter). We understand by location of a user the IS the user is registered with. The behaviour of the *Loc* ambient and the *Utest* ambient are specified exactly with appropriate initial values. The ISC requests the value of any of these cells by sending the name *Loc* or *Utest* to the user's account ambient (see Eq. (14)) which then can *get* (i.e. read) the value of the corresponding child ambient as follows, where the parameter x is the corresponding child ambient name:

$$!\,ISC \uparrow (x).x \downarrow \langle\rangle.x \downarrow (y).ISC \uparrow \langle y\rangle.0 \tag{11}$$

The user's account ambient can also put (i.e. write) a value in any of its child ambients as follows, where the parameter x is the corresponding child ambient name and the parameter n is that value:

$$!\,ISC \uparrow (x,n).x \downarrow \langle n\rangle.x \downarrow (\,).ISC \uparrow \langle x, ACK\rangle.0 \tag{12}$$

So the whole behaviour of a user's account ambient named uid is specified as:

$$P_{uid} \triangleq Eq.(11) \mid Eq.(12)$$

Lectures As mentioned above, this ambient contains all the lectures that are available in the mLecture service. Each lecture has a unique ID and the corresponding ambient is named

after that ID. The behaviour of a lecture ambient is specified in Eq. (7). The *Lectures* ambient behaves exactly as the cache ambient specified in Eq. (6).

Infostation centre We now formalise the behaviour of the ISC when it receives a request from an IS. We are interested in two types of request in this paper: an AAA request and a lecture request.

When the ISC receives an AAA request from an IS, it updates the user's account with its new location and then replies to that IS with an acknowledge along with a list of available services. For the sake of simplicity, the service list is represented by the name '*SLIST*'. This is modelled by the following process:

$$!\,IS_i :: (aaareq, uid, dtype, aname).uid \downarrow \langle IS_i \rangle.uid \downarrow (ack). \qquad (13)$$

$$IS_i :: \langle ack, aname, SLIST \rangle.0$$

After receiving a lecture request, the ISC checks whether the user requesting the service is currently taking a mTest. This is done by it sending a message to the corresponding user's account ambient. That user's account ambient reply with 0 for 'No' and 1 for 'Yes'. If the reply is 'Yes' then the ISC fetches the current location of the user and forward a 'DENIED' message to that location. If the reply is 'No' and the flag is set to 1, a 'OK' message is forwarded to the current user location; otherwise (i.e. reply is 'No' and the flag is set to 0), the ISC fetches the appropriate lecture version for the user device and sends it to the current user location. This behaviour is represented by the following process:

$$!\,IS_i :: (lectid, uid, dtype, aname, flag).uid \downarrow \langle Utest \rangle.uid \downarrow (y).\left(P_{ISC_1} \mid P_{ISC_2}\right) \qquad (14)$$

where

$$P_{ISC_1} \triangleq (y = 1)?\,uid \downarrow \langle Loc \rangle.uid \downarrow (z).z :: \langle lectid, DENIED, aname \rangle.0$$

and

$$P_{ISC2} \triangleq \left(\begin{array}{l} (y = 0 \wedge flag = 1)?uid\downarrow\langle Loc\rangle.uid\downarrow(z). \\ z :: \langle lectid, OK, aname\rangle.\mathbf{0} \mid \\ (y = 0 \wedge flag = 0)?Lectures\downarrow\langle lectid, uid, \\ dtype, aname\rangle.Lectures\downarrow(lectid, uid, \\ dtype, reply, aname).uid\downarrow\langle Loc\rangle.uid\downarrow(z). \\ z :: \langle lectid, reply, aname\rangle.\mathbf{0} \end{array} \right)$$

So the whole behaviour of the ISC is modelled by the following process:

$$P_{ISC} \triangleq Eq.(13) \mid Eq.(14)$$

7. Validation

Now that a formal model of the infostation-based mLearning system has been presented, we show how this model can be used to validate the properties of the mLearning system. Lamport proposed two main classes of system's properties: *safety properties*, which state that

'nothing bad will happen'; and *liveness properties*, which assert that 'something good will happen, eventually'. In the light of this classification, we wish to establish the liveness property that every lecture request from a user is eventually replied to by the system, provided that the user does not become infinitely unavailable after that request has been made.

Theorem 7.1 *Every user's lecture request will eventually get a reply, provided that the user stays long enough in the system.*

The proof of this theorem is based on the reduction semantics of given by a congruence relation '≡' defined in Table 9 and a reduction relation '→' defined in Table 10.

In this proof we will assume, without loss of generality, that the user is using a laptop (PC) to access the system from an infostation IS_i. Given that the user is mobile, the following cases must be considered:

1. the user sends the request and waits for the reply in the same infostation IS_i (i.e. the user may move around within the range of the infostation)
2. the user sends the request and move into a different infostation $IS_j, j \neq i$.

In Case 1, the user behaviours can be modelled by the following ambient:

$$PC303[Lectreq_i :: \langle Lect001, 303, PC, PC303\rangle.Lectreq_i :: (lectid, reply).0] \quad (15)$$

This ambient sends a lecture request to the $Lectreq_i$ ambient and waits for a reply from that ambient, and then terminates. The lecture request contains the following information: (i) the lecture ID, *Lect001* ; (ii) the user ID, *303*; (iii) the device type, *PC*; and (iv) the name of the ambient to reply to, *PC303*.

The behaviours of the $Lectreq_i$ ambient is modelled by the process P_{L_i} in Eq. (5), which basically receives a lecture request from a sibling ambient (e.g. a user device), forward the request to the infostationIS_i, get the reply from that infostation and forwards it to the very ambient which initiated the request. How the infostation IS_i interacts with the $Lectreq_i$ ambient is specified in Eq. (8). These interactions between a user device, the $Lectreq_i$ ambient and the infostation IS_i can be expressed as a sequence of derivations using the reduction relation →. Because of the space limit, we cannot give the full sequence of derivations in this paper. For illustration, the following sequence of derivations describes how a lecture request sent by the user gets to the infostation to be processed:

$$\begin{aligned}
&\text{Eq. (8)} \mid \text{Eq. (15)} \mid Lectreq_i[P_{L_i}] \\
\rightarrow\ &\{\text{Rule (Red Com S2) in Table 10; the request is sent to the } Lectreq_i \text{ ambient}\} \\
&\text{Eq. (8)} \mid PC303[Lectreq_i :: (lectid, reply).\mathbf{0}] \mid \\
&Lectreq_i \left[\begin{array}{l} IS_i \uparrow \langle Lect001, 303, PC, PC303\rangle.\mathbf{0} \mid \\ ! :: (lectid, uid, dtype, aname). \\ IS_i \uparrow \langle lectid, uid, dtype, aname\rangle.\mathbf{0} \mid \\ !IS_i \uparrow (lectid, reply, aname). \\ aname :: \langle lectid, reply\rangle.\mathbf{0} \end{array}\right]
\end{aligned}$$

$\rightarrow$ {Rule (Red Com R6) in Table 10; the $Lectreq_i$ ambient forwards the request to the infostation}

$$\text{Eq. (8)} \mid Cache_i\downarrow\langle Lect001, 303, PC, PC303\rangle.\mathbf{0} \mid PC303[Lectreq_i :: (lectid, reply).\mathbf{0}] \mid Lectreq_i \left[\begin{array}{l} ! :: (lectid, uid, dtype, aname). \\ IS_i \uparrow\langle lectid, uid, dtype, aname\rangle.\mathbf{0} \mid \\ !IS_i \uparrow(lectid, reply, aname). \\ aname :: \langle lectid, reply\rangle.\mathbf{0} \end{array}\right]$$

At this stage, the infostation IS_i has received a lecture request from the $Lectreq_i$ ambient and is willing to check with the $Cache_i$ ambient whether it has the requested lecture for the specified type of device. The behaviour of the $Cache_i$ ambient is specified by the process P_{C_i} in Eq. (6). If the requested lecture *Lect001* exists in the cache for the specified type of device, then the $Cache_i$ ambient gets a copy of the lecture for the specified type of device by interacting with the child ambient named *Lect001* whose behaviour is specified by the process $P_{Lect001}$ as in Eq. (7). It can also be seen from Eq. (6) and Eq. (7) that if the requested lecture *Lect001* does not exist in the cache for the specified type of device, then a reply message 'NULL' is forwarded to the infostation IS_i. So in either situation, the infostation IS_i receives a reply from the cache.

Once a reply is received from the $Cache_i$ ambient, the infostation IS_i contacts the infostation centre *ISC* as specified by the process in Eq. (9). How the *ISC* reacts is modelled by Eq. (14); it replies with a 'DENIED' message if the user requesting the lecture is currently using a mTest service, otherwise it replies with a 'OK' message and possibly a copy of the requested lecture if it is not available locally in IS_i's cache. How each of these types of reply is handled by the IS_i is modelled by the component *C* and *N* in Eq. (9). One can see that for every case where the user is still in the range of the infostation IS_i (i.e. the context expression 'has(*aname*)' holds), the infostation IS_i sends a reply to the $Lectreq_i$ ambient which subsequently forwards the reply to the user device as specified in Eq. (4). This completes the proof of Case 1.

The proof of Case 2 can be done in a similar manner as in Case 1, with the user behaviours specified as in Eq. (16), where $i \neq j$, i.e. the request is sent from one infostation and the reply to that request is received after the user has moved to a different infostation.

$$PC303 \left[\begin{array}{l} Lectreq_i :: \langle Lect001, 303, PC, PC303\rangle.\mathtt{out}. \\ \mathtt{in}\ IS_j.AAAreq_j :: \langle 303, PC, PC303\rangle.\mathbf{0} \mid \\ \mathtt{at}(IS_j)?AAAreq_j :: (ack, slist). \\ Lectreq_j :: (lectid, reply).\mathbf{0} \end{array}\right] \quad (7)$$

This ambient sends a lecture request from the infostation IS_i, moves to a different infostation IS_j , registers with this infostation by sending an AAA request then waits for the acknowledgement of its registration. Once its registration has been confirmed, it then prompts to receive the reply to the lecture request and then terminates.

8. Acknowledgment

The authors wish to acknowledge the support of the National Science Fund (Research Project Ref. No. DO02-149/2008) and the Science Fund of the University of Plovdiv "Paisij Hilendarski" (Research Project Ref. No. NI11-FMI-004).

9. References

[Barker,2000] P. Barker, Designing Teaching Webs: Advantages, Problems and Pitfalls, in Proc. of ED-MEDIA 2001 World Conference on Educational Multimedia, Hypermedia Telecommunication, Association for the Advancement of Computing in Education, Charlottesville, VA, 2000, pp. 54-59.

[Bellifemine,2007] F. Bellifemine, G. Caire, D. Greenwood, Developing Multi-Agent Systems with JADE, John Wiley & Sons Ltd., 2007.

[Bucur,2008] D. Bucur, M. Nielsen, Secure Data Flow in a Calculus for Context Awareness, in: Concurrency, Graphs and Models, Vol. 5065 of Lecture Notes in Computer Science, Springer, 2008, pp. 439-456.

[Bugliesi,2004] M. Bugliesi, G. Castagna, S. Crafa, Access Control for Mobile Agents: The Calculus of Boxed Ambients, ACM Trans. on Programming Languages and Systems, 26 (1), 2004, 57-124.

[Cardelli,2000] L. Cardelli, A. Gordon, Mobile Ambients, Theoretical Computer Science 240, 2000, 177-213.

[Dey,2000] Dey, A.K., Abowd, G.D.Towards a better understanding of context and context-awareness. Proceedings of the Workshop on the What, Who, Where, When and How of Context-Awareness, New York, ACM Press, 2000.

[FIPA,2002] FIPA, ACL Message Structure Specification, Foundation for Intelligent Physical Agents, Geneva, Switzerland SC00061G, 3rd December 2002.

[Frenkiel,1996] R. Frenkiel and T. Imielinski, Infostations: The joy of `many-time, many-where' communications, WINLAB Technical Report,1996.

[Ganchev, 2005] Ganchev, I., S. Stojanov, M. O'Droma. Mobile Distributed e-Learning Center. In Proc. of the 5th IEEE International Conference on Advanced Learning Technologies (IEEE ICALT'05), pp. 593-594, Kaohsiung, Taiwan. DOI 10.1109/ICALT.2005.199. ISBN 0-7695-2338-2. 5-8 July 2005.

[Ganchev,2007] I. Ganchev, et al., An InfoStation-Based Multi-Agent System Supporting Intelligent Mobile Services Across a University Campus, Journal of Computers, vol. 2, pp. 21-33, May 2007.

[Ganchev,2008a] I. Ganchev, et al., On InfoStation-Based Mobile Services Support for Library Information Systems, in 8th IEEE International Conference on Advanced Learning Technologies (IEEE ICALT-08), Santander, Cantabria, Spain, 2008, pp. 679 - 668.

[Ganchev,2008b] I. Ganchev, et al., InfoStation-Based Adaptable Provision of m-Learning Services: Main Scenarios, International Journal Information Technologies and Knowledge (IJ ITK), vol. 2, pp. 475-482, 2008.

[Ganchev,2008c] I. Ganchev, et al., InfoStation-based mLearning System Architectures: Some Development Aspects, in 8th IEEE International Conference on Advanced Learning Technologies, (ICALT'08), Santander, Spain, 2008, pp. 504-505.

[Maurer,2001] H. Maurer and M. Sapper, E-Learning Has to be Seen as Part of General Knowledge Management, in Proc. of ED-MEDIA 2001 World Conference on Educational Multimedia, Hypermedia Telecommunications, Tampere, AACE, Chalottesville, VA, 2001, pp. 1249-1253.

[Milner,1999] R. Milner. Communication and Mobile Systems: The π-Calculus. Cambridge University Press, 1999.

[O'Droma,2007] M. O'Droma and I. Ganchev, Toward a Ubiquitous Consumer Wireless World, IEEE Wireless Communications, vol. 14, pp. 52-63, February 2007.

[OWL-S,2010] OWL-S: Semantic Markup for Web Services. [Online]. Available - http://www.w3.org/Submission/OWL-S/ [Accessed: Mar 3, 2010].

[Passas,2006] N. Passas, et al., Enabling technologies for the 'always best connected' concept: Research Articles, Wirel. Commun. Mob. Comput., vol. 6, pp. 523-540, 2006.

[Sangiorgi,2001] D. Sangiorgi and D. Walker. The π-calculus: A Theory of Mobile Processes. Cambridge University Press, 2001.

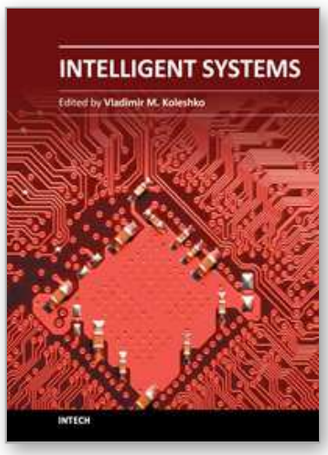
[Stoyanov,2005] S. Stoyanov, et al., From CBT to e-Learning, Journal Information Technologies and Control, vol. 4, pp. 2-10, 2005.

[Stoyanov,2008] S. Stoyanov, et al., An Approach for the Development of InfoStation-Based eLearning Architectures Compt. Rend. Acad. Bulg. Sci., vol.61, pp. 1189-1198, 2008.

[W3C,2010] W3C, Document Object Model (DOM) [online]. Available - http://www.w3.org/DOM/. [Accessed Mar 03, 2010].

[Zedan,2008] H. Zedan, A. Cau, K. Buss, S. Westendorf, A. Hugill, S. Thomas, Mapping Human Creativity, STRL Internal Monograph, STRL-2008-09, De Montfort University, Leicester, 2008,UK.

[Zimmer,2005] P. Zimmer, A Calculus for Context-awareness, Tech. rep., BRICS, 2005.



Intelligent Systems

Edited by Prof. Vladimir M. Koleshko

ISBN 978-953-51-0054-6
Hard cover, 366 pages
Publisher InTech
Published online 02, March, 2012
Published in print edition March, 2012

This book is dedicated to intelligent systems of broad-spectrum application, such as personal and social biosafety or use of intelligent sensory micro-nanosystems such as "e-nose", "e-tongue" and "e-eye". In addition to that, effective acquiring information, knowledge management and improved knowledge transfer in any media, as well as modeling its information content using meta-and hyper heuristics and semantic reasoning all benefit from the systems covered in this book. Intelligent systems can also be applied in education and generating the intelligent distributed eLearning architecture, as well as in a large number of technical fields, such as industrial design, manufacturing and utilization, e.g., in precision agriculture, cartography, electric power distribution systems, intelligent building management systems, drilling operations etc. Furthermore, decision making using fuzzy logic models, computational recognition of comprehension uncertainty and the joint synthesis of goals and means of intelligent behavior biosystems, as well as diagnostic and human support in the healthcare environment have also been made easier.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

S. Stoyanov, H. Zedan, E. Doychev, V. Valkanov, I. Popchev, G. Cholakov and M. Sandalski (2012). Intelligent Distributed eLearning Architecture, Intelligent Systems, Prof. Vladimir M. Koleshko (Ed.), ISBN: 978-953-51-0054-6, InTech, Available from: http://www.intechopen.com/books/intelligent-systems/provably-correct-intelligent-elearning-environment

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen