Secure and Dynamic Client Environment for Interactive eHomeCare

Sofie Van Hoecke[1], Kristof Taveirne[1], Stein Desmet[1], Brecht Claerhout[2], Filip De Turck[1], Bart Dhoedt[1]

[1]Department of Information Technology, IBCN - IBBT - IMEC, Ghent University
Gaston Crommenlaan 8 bus 201, Ghent, Belgium
sofie.vanhoecke@intec.ugent.be
[2]Custodix NV, Verlorenbroodstraat 120/14, Merelbeke, Belgium

Keywords: eHomeCare, dynamic client, user friendliness, Eclipse RCP, Single-Sign-On

Abstract— **Ehomecare services can save money and improve patient care due to increased feelings of well-being at home. However, since not all patients are technical experts and ehomecare services are developed by different vendors, using different languages and data definitions, acceptance of ehomecare might become difficult due to the complexity of managing all the different service locations and login information.**
Therefore the authors present in this paper a patient centered application minimizing user interactions and simplifying authentication. The client application is adjusted dynamically to fit the patient profile, fulfilling the specific needs for each patient personally and presents only the ehomecare services that are of interest to the patient. The patient only needs to authenticate once using his eID at login time, and afterwards the platform provides single-sign-on to the different ehomecare services.

I. Introduction

The ageing population and a shift in the burden of illness from acute (e.g. infections and injury) to chronic conditions (e.g. heart disease, asthma, epilepsy, cancer, mental health problems, etc.) drive up health costs as well as the need for healthcare services, while there are fewer people to provide these services. Patients are discharged from hospitals earlier, to save money and to improve patient care due to increased feelings of well-being at home. It is however not permitted for them to go home, unless the home is fitted with the appropriate technology, requiring additional healthcare services and monitoring of their health status. Ehomecare services like telemonitoring can provide cost-effective means to save valuable healthcare service time by monitoring body functions from a distance. Besides telemonitoring, ehomecare requires sharing of responsibility, services (like escheduling, eprescription, alarmservice) and decision making.
Integration of ehomecare services and applications is a complex task since these services and applications are built by different vendors, and use different data definitions and exchange standards. The IBBT-Coplintho project [1] investigated how an ehomecare environment could be designed and developed in order to support the care process for patients treated at home. All ehomecare services, currently available within Coplintho and to be developed in the future, can be integrated in a multi-functional platform, fulfilling patient's requirements. A Web service broker has been implemented and was presented in [2] which selects and composes ehomecare services for fulfilling the user request. On top of that, some example end-to-end services and applications such as escheduling, telemonitoring, emergency call, audio diary and video conferencing, have been developed to illustrate the advantages of service integration and composition for ehomecare. In [3] the authors presented an agent-based platform for efficient processing of the telemonitoring data retrieved by the Coplintho platform. Software agents support the physician through medical data processing of the patient's vital signs measurements such as weight, glucose level, blood pressure, blood oxygen level, pulse and/or ECG rhythm. With the Interactive Patient Terminal, a patient can transmit these vital signs data to the Coplintho platform. However, ease-of-use, user-friendliness and security of the Interactive Patient Terminal at home are required in order to guarantee the acceptance and penetration of new modes of practice in the field of ehomecare, and has been the subject of recent research. Service provisioning and user interaction should be as simplified as possible since not all ehomecare users and providers are technical experts.
Therefore, in this paper, the authors present a patient centered application, running on the Interactive Patient Terminal, as a portal to the Coplintho broker platform, this way minimizing user interactions and simplifying authentication. The presented client application is adjusted dynamically to fit the patient profile, fulfilling the specific needs for each patient personally. The patient only authenticates once using his eID at login time, and afterwards the platform provides single-sign-on to the different ehomecare services.
The remainder of this paper is structured as follows: Section II describes our dynamic client environment, while in section III single-sign-on is presented. Finally, in section IV, we will highlight the main conclusions and identify some future work.

II. Dynamic Client Environment

Within ehomecare, each patient has his specific needs for optimal personal healthcare, depending on his pathology. A patient

is not interested in ehomecare services for other pathologies. Therefore we created a dynamic client environment in which the patient, after authenticating, only has the services available that are of interest to him.
The main challenge is to create an application which is very easy to use and has a quick learning curve. Therefore a graphical subset of the Eclipse Rich Client Platform (RCP) [4] was created on which the Coplintho service providers can develop plug-ins. The Eclipse RCP provides a shell of an application, along with a module-based API that enables you to build your own application on top of this shell. Eclipse RCP creates a good design for adding menus, toolbars and different views. The RCP plug-ins can be written in Java using the Eclipse Plugin Development Environment (PDE) [5]. However non-Java plug-ins are also possible such as a Windows Media Player or Internet Explorer plug-in or .NET applications provided as ActiveX. Navigation in between plug-ins is implemented using the navigation plug-in.
The Eclipse RCP plug-ins are in fact Open Standard Gateway Initiative (OSGI) bundles combined with the graphical features of the Eclipse IDE. Since OSGi is a dynamic component model, the Eclipse plug-ins can be added to the client application without having to reboot the application.

A. *Loose coupling*

The core of the client side application is a reusable piece of software, following the Microkernel design pattern. The Microkernel design pattern applies to software systems that must be able to adapt to changing system requirements. It separates a minimal functional core from extended functionality and application-specific parts. In the Coplintho setting, the Microkernel's minimal functionality must take the form of simple management operations, i.e. starting, initializing, authentication, combining and stopping of modules as well as dispatching of messages between them. This approach requires modules to be uniform, deployable and loosely coupled so that they can be plugged into the client core. The core of the application uses different modules to become a fully operational application. This way the modules may be replaced by other implementations at a later stage. As an example, one of these modules can implement the authentication mechanism. Where Coplintho currently uses the Belgian eID to provide identity information, one could replace that module with another one using a wizard to ask the user to input his identity.
The different modules are grouped in a feature. A feature is nothing more than a collection of Eclipse RCP plug-ins that are meant to be installed as a group. A feature can be updated as a whole or plug-ins can be updated individually as long as the plug-in's version is within the restrictions specified in the feature.

B. *Coplintho feature*

The Coplintho feature (see Figure 1) contains all Coplintho specific implementations of the different modules needed by the Microkernel. From then on, the application core uses the modules of the feature to achieve the needed functionality.

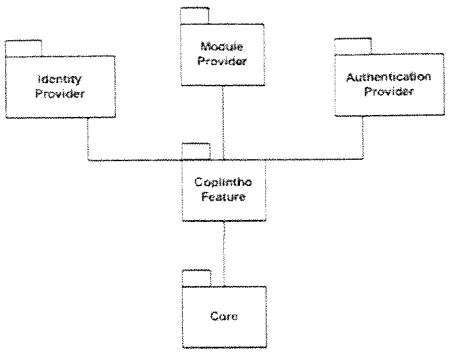

Fig. 1. The client's core using the Coplintho feature

Identity provider
Within Coplintho, the identity of the user is read from the Belgian eID card. The eID is a legal identity card which allows the holder to digitally sign documents and authenticate himself to websites or other applications. All code necessary for reading out the eID is inside a plug-in of the Coplintho feature and uses the eID middleware provided by Belgium's federal government.

Module provider
The modules are provided to the application using an online repository. A configuration Web service is responsible for the configuration of the client application the user is logged in to. Since the configuration is bound to the user, all non-feature modules specific to that user are removed from the system when the patient removes his eID card from the reader.

Authentication provider
Since Coplintho is a Web service framework, brokering services from different providers, single-sign-on is needed to alleviate users from authenticating to each service separately. The authentication provider provides the core with long term and short term SAML assertions obtained from the Coplintho trusted security provider.

C. *Non-feature modules*

In order to provide loose coupling, Eclipse RCP uses the mechanism of extensions and extension points. When a plug-in wants to allow other plug-ins to extend or customize portions of its functionality, it will declare an extension point. The extension point declares a contract that extensions must conform to. Plug-ins that want to connect to that extension point must implement that contract in their extension without the plug-in being extended knowing about it. This allows plug-ins built by different service providers within Coplintho to interact seamlessly, even without knowing details about one another.
This way service providers can create modules provided to the Coplintho platform by extending Eclipse RCP plug-ins in

the plugin.xml file following the below extension schema:

```
<element name="configuration">
  <complexType>
    <sequence>
       <element ref="relatedPlugin"/>
       <element name="documentation" ...>
       </element>
    </sequence>
    <attribute name="perspectiveId" ...>
    </attribute>
    <attribute name="title" ...>
    </attribute>
    <attribute name="image" ...>
    </attribute>
    <attribute name="category" ...>
    </attribute>
    <attribute name="launchAtInstall" ...>
    </attribute>
    <attribute name="manual" ...>
    </attribute>
  </complexType>
</element>
<element name="relatedPlugin">
  <complexType>
    <attribute name="id" ...>
    </attribute>
  </complexType>
</element>
```

By following the above schema, a set of buttons for the module is added to the toolbar. This way easy navigation is enabled to the different modules with access to the documentation of the modules.
Currently implemented modules by service providers are escheduling, video conferencing, audio diary, telemonitoring, emergency alarm, pathology information and medication reminder.

D. Remote configuration

Since the need of a patient may change in time, the module configuration needs to be remotely configurable. At regular times, the client application checks a configuration Web service. If the configuration has changed due to a new version of a module or a newly added module, an update will take place. This module will then be downloaded and installed at runtime, without any required effort from the patient. Administration of the configurations can only be done by privileged users such as patient's doctors. This way, privileged users can remotely activate and/or deactivate selected modules for a specific user based on his pathology, requirements or profile. The client application is dynamically adjusted to this configuration without any effort of the patient.

E. Using the client environment

When a user enters the Coplintho application, he is requested to place his eID into the card reader and enter his secret code (Figure 2). As can be seen in the figure, besides the feature plug-ins, no other plug-ins, specific for the patient's profile, are installed yet.

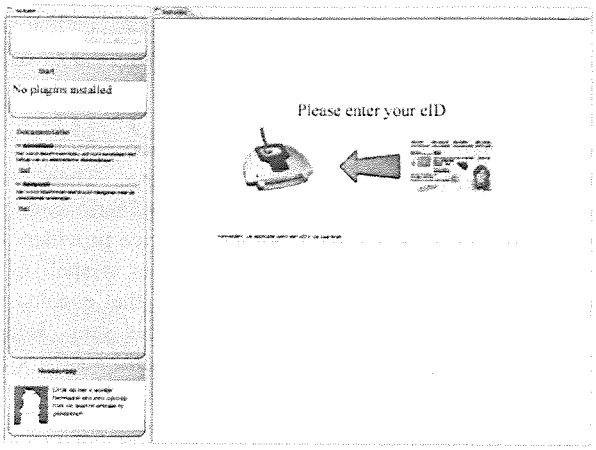

Fig. 2. Starting the Coplintho application

Once authenticated, a set of buttons is added to the toolbar, giving access to the patient specific modules, such as video conferencing, audio diary and telemonitoring (Figure 3).

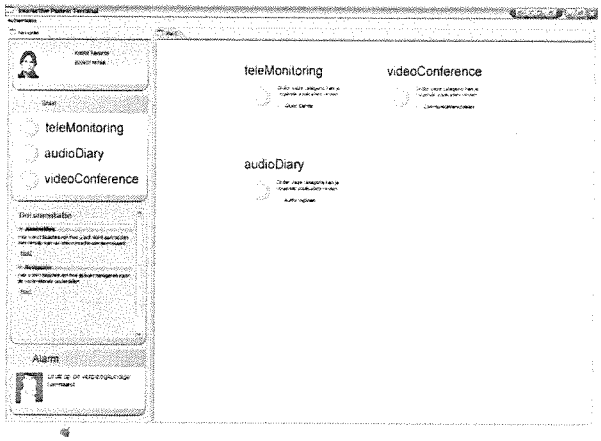

Fig. 3. Coplintho application extended with patient specific modules such as teleMonitoring, audioDiary and videoConference

The patient can now use the modules specific for his profile and/or pathology by navigating the toolbar and activating the according module (Figure 4) without being asked to authenticate again for each module.
Since each of these service provider modules requires its own authentication, single-sign-on is used to alleviate users from authenticating to each service separately. Single-sign-on is presented more in detail in the next section.

III. Single-Sign-On

The idea behind Single-sign-on (SSO) is that the user authenticates once and is then automatically authenticated to use all available services in a secure way. Within Coplintho, we support two kinds of SSO, namely Web SSO and SSO over Web services, where Web SSO means that once authenticated to a website, the user gains access to all other

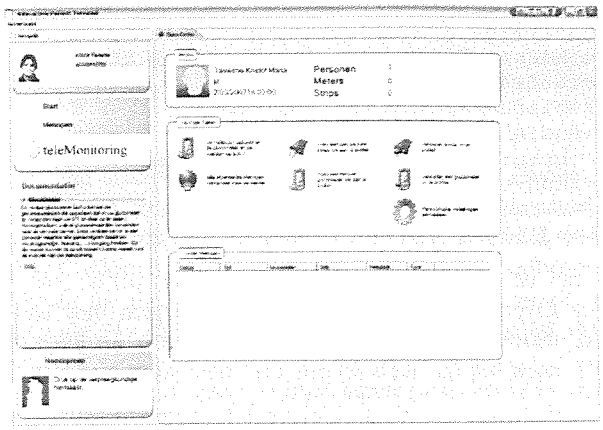

Fig. 4. TeleMonitoring module opened inside the Coplintho application

websites that need authentication.
In order to implement SSO within Coplintho, an Authentication Web Service (AWS) is implemented. Once authenticated to this AWS, assertions can be asked as tokens to prove authentication to the different modules. The modules can check these assertions on correctness and if they are originating from the trusted AWS.

A. Assertions

SAML (Security Assertion Markup Language) [6] is an OASIS standard defining a framework for exchanging security information between entities. The Coplintho AWS is a SAML authority that returns Long Lived or Short Lived SAML assertions. A SAML authority is an Identity Provider that asserts information about a subject. Long Lived SAML assertions are used to authenticate the client to the AWS, while Short Lived SAML assertions can be presented to the modules requiring authentication. It is up to the service providers of the modules whether to trust these assertions from the AWS or not.
WS-Trust [7] describes a protocol to use a SignChallenge to set up the required trust relation. Since open source implementations of WS-Trust were premature at the time of development, the handshake protocol needed for SSO was not yet incorporated and the AWS needed to be developed from scratch using SAAJ [8] and XWS-Security [9]. More recently a lot of effort has been invested in the XWSS [10] and WSIT [11] projects providing the required functionality.
The claimant needs to send a RequestSecurityToken to the AWS, holding the type of security token that is requested (Figure 5 - 1).

```
<wst:RequestSecurityToken>
<wst:TokenType>
urn:oasis...WSS:1.0:profiles:WSS-SAML-profile
</wst:TokenType>
<wst:RequestType>
http://schemas.xmlsoap.org/ws/.../trust/Issue
</wst:RequestType>
</wst:RequestSecurityToken>
```

The AWS does however not trust the timestamp in this message and sends a SignChallenge back to the claimant (Figure 5 - 2). Since the specification does not specify the format of this challenge, the FIPS PUB 196 schema is used here. FIPS PUB 196 is a standard that defines two protocols for entity authentication using public key cryptographic algorithms for generating and verifying digital signatures. The first one is used here in order to let an entity prove its identity by using a private key to generate a digital signature on a random challenge (Figure 5 - 3).

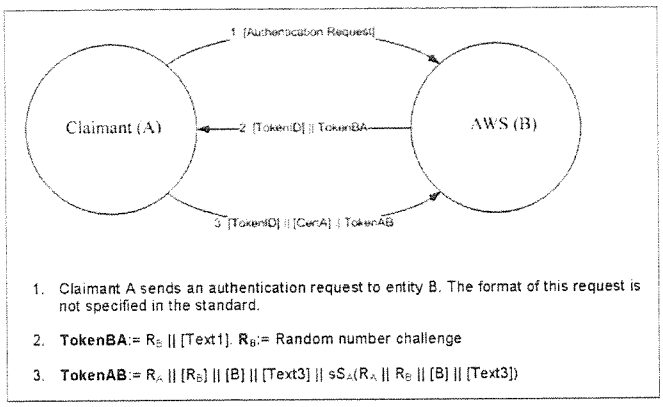

Fig. 5. Authentication between claimant and AWS using the FIPS PUB 196 schema

Since the FIPS PUB 196 standard does not specify the format of the authentication request token. The following schema is used to implement this:

```
<schema targetNamespace="urn:coplintho:fips196"
 xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:cdx-fips196="urn:coplintho:fips196"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified"
 version="1.1">
<element name="VerifierChallenge"
 type="string"/>
<element name="ClaimantChallenge"
 type="cdx-fips196:ClaimantChallengeType"/>
<complexType name="ClaimantChallengeType">
<complexContent>
<sequence>
<element name="GeneratedChallenge"
 type="string"/>
<element name="Response" type="string"/>
</sequence>
</complexContent>
</complexType>
</schema>
```

This results in the following SignChallenge of the AWS to the claimant:

```
<wst:RequestSecurityTokenResponse>
<SignChallenge>
<Challenge>
    <fips196:VerifierChallenge
xmlns:fips196="urn:custodix:coplintho:fips196">
85728109664837998436066035344297346966652640...
```

```
    </fips196:VerifierChallenge>
</Challenge>
</SignChallenge>
</wst:RequestSecurityTokenResponse>
```

The claimant reacts on the SignChallenge by concatenating a random number to the challenge and sends this back to the AWS in a signed SignChallengeResponse block.

```
<wst:RequestSecurityTokenResponse>
<SignChallengeResponse>
<Challenge>
    <fips196:ClaimantChallenge
xmlns:fips196="urn:custodix:coplintho:fips196">
        <fips196:GeneratedChallenge>
2320251463350103513481543351736135664370068...
        </fips196:GeneratedChallenge>
        <fips196:Response>
2320251463350103513481543351736135664370068...
        </fips196:Response>
    </fips196:ClaimantChallenge>
</Challenge>
</SignChallengeResponse>
</wst:RequestSecurityTokenResponse>
```

The AWS checks if the challenge is indeed the challenge that was sent to the claimant and if the signature is verified. As a result, the AWS will send a Long Lived SAML Authentication Assertion back to the claimant. As can be seen below, the SAML Authentication Assertion has the AWS itself as its audience. The lifetime of the Long Lived assertion is specified in the Conditions xml tag. During this lifetime, the handshake protocol does not need to be repeated and the assertion can be used to ask the AWS for Short Lived assertions.

```
<Conditions
 NotBefore="2007-03-02T12:25:07.734Z"
 NotOnOrAfter="2007-03-02T14:25:07.734Z">
 <AudienceRestrictionCondition>
  <Audience>
   http://aws.coplintho.be/
  </Audience>
 </AudienceRestrictionCondition>
</Conditions>
```

To obtain a Short Lived SAML Assertion a request is sent to the AWS.

```
<wst:RequestSecurityToken ...>
...
<wsp:AppliesTo ...>
<wsa:EndpointReference ...>
 <Address>
 http://coplintho.ibbt.be/broker
 </Address>
 <ServiceName>Broker</ServiceName>
</wsa:EndpointReference>
</wsp:AppliesTo>
</wst:RequestSecurityToken>
```

In the header of this request the Long Lived Assertion is added. If the Long Lived assertion is still valid, the AWS has proof that the client has been authenticated before and the AWS can fulfill the client's request by sending the Short Lived SAML Assertion it was asked for.

```
<Conditions
 NotBefore="2007-03-02T12:25:08.859Z"
 NotOnOrAfter="200-03-02T12:25:18.859Z">
 <AudienceRestrictionCondition>
  <Audience>
    http://coplintho.ibbt.be/broker
  </Audience>
 </AudienceRestrictionCondition>
</Conditions>
```

As can be seen, the Short Lived assertions differ from the Long Lived assertions by the period in which they can be used, which is much shorter. The audience is no longer the AWS itself but the Web service the client wishes to talk to.

B. Delegation

The domain analysis within the Coplintho project and multiple interactive discussions between the actors involved in the project (nurses, general practitioners, hospital physicians and specialists) highlighted the need for integration and composed advanced ehomecare services crossing healthcare providers and medical databases. Therefore a Coplintho broker platform [2] was developed to achieve this integration and composition. Dynamically selecting and composing ehomecare services, automatically load balanced and with guaranteed QoS, will result in many new and advanced ehomecare services offering functionality currently not achievable. The Coplintho broker architecture thereby facilitates and simplifies providing services to the framework. Service providers only need to provide a Web service interface on their applications, register them transparently to the framework and the platform takes care of all the rest.

Using a broker middleware however impacts single-sign-on principles since the client application only communicates with the Coplintho broker. The broker then takes care of all requests done by the client, selects and composes needed services and forwards the requests to the selected service providers. In order to provide single-sign-on, the client should delegate the right to access those services to the broker. Therefore the broker must have proof that it is delegated by the client. WS-Trust specifies a delegation syntax supporting this, where the DelegateTo element should specify the X509 certificate of the broker.

```
<wst:RequestSecurityToken ...>
 <TokenType>
   urn:oasis:...:profiles:WSS-SAML-profile
 </TokenType>
 <RequestType>
   http://schemas.xmlsoap.org/ws...trust/Issue
 </RequestType>
 <wst:DelegateTo>
    <wsse:BinarySecurityToken>
      ...
    </wsse:BinarySecurityToken>
 </wst:DelegateTo>
</wst:RequestSecurityToken>
```

The client asks a Long Lived SAML Delegation Assertion to the AWS and sends the Long Lived SAML Authentication Assertion in the header. The broker can use this Long Lived

SAML Delegation Assertion to ask the AWS for Short Lived SAML Delegation Assertions in order to be able to invoke the services for the client.

C. Web SSO

Using the assertions, the platform is able to invoke the Web services provided by the service providers. However for example the pathology information service is a website created within Coplintho where depending on the login, information is shown for the user's pathology. Therefore we would also like to use the assertions to gain access to the websites within Coplintho.

A Shibboleth [12] server is used server side for Web single-sign-on (Web SSO). Shibboleth is an open, standards-based solution to exchange information about users in a secure, and privacy-preserving manner and provides Web SSO across or within organizational boundaries.

In order to transfer the assertions to this Shibboleth server, the client passes the assertion as a hidden form field in generated HTML code. The HTML is passed to a browser which submits the form to the Shibboleth server. The assertions are then analyzed server side and a Web SSO is started.

By using a combination of SSO over Web services and Web SSO, the assertions can be used to authenticate the user to the different Web services of the service providers, as well as to the websites provided within Coplintho.

IV. Conclusions and Future Work

In order to save money and increase the patient's feelings of well-being at home, patients are discharged earlier and need to rely on ehomecare services monitoring their health and assisting them where needed. However, since not all patients are technical experts and ehomecare services are developed by different service providers, usability and acceptance of ehomecare might become difficult due to the complexity of managing all the different service locations and login information.

Therefore the authors presented in this paper a patient centered application minimizing user interactions and simplifying authentication. Privileged users can configure patients' profiles and activate and/or deactivate available modules for each patient depending on their needs, pathology or profile. The client application is adjusted dynamically to fit the patient profile, without any required interaction of the patient and fulfilling the specific needs for each patient personally by presenting only the ehomecare services that are of interest to the patient.

Since each of the ehomecare service provider modules requires its own authentication, single-sign-on is used to alleviate users from authenticating to each service separately.

Since integration and combination of ehomecare services will result in many new and advanced ehomecare services offering functionality currently not achievable, a Coplintho broker platform was developed to achieve this integration and composition. The client application can also interact with the Coplintho broker and delegates his right to access the ehomecare services to the broker using delegation assertions. By using a combination of SSO over Web services and Web SSO, the assertions and delegation assertions can be used to authenticate the user to the different Web services of the service providers, as well as to the websites provided within Coplintho.

We will continue the design of our dynamic client environment by creating policies and implementing access control for remote configuration of the patient's profile through emerging Web service standards such as XACML [13].

V. Acknowledgment

Coplintho is a project of IBBT (Interdisciplinary institute for BroadBand Technology) in cooperation with the following companies and organizations: WGK, Televic, MediBRIDGE, Androme, Custodix, UZGent, Sint-Elisabeth Ziekenhuis Zottegem. IBBT is a research institute founded by the Flemish Government in 2004.

Sofie Van Hoecke would like to thank the IWT (Institute for the Promotion of Innovation through Science and Technology in Flanders) for financial support through her Ph.D. grant.

Filip De Turck acknowledges the F.W.O.-V. (Fund for Scientific Research-Flanders) for their support through a postdoctoral fellowship.

References

[1] A. Ackaert, S. Van Hoecke, G. De Moor, L. Spinhof, S. De Rouck, S. Agten, P. Verhoeve, Innovative Communication Platforms for Interactive eHomeCare, Invited Paper, The 5th IEEE workshop on Application and Services in Wireless Networks (ASWN05), Paris, France, 2005.

[2] S. Van Hoecke, K. Vlaeminck, F. De Turck, B. Dhoedt, Open Web Services-based Middleware for Brokering of Composed eHomeCare Services, 2005 Middleware for Web Services (MWS'05) Workshop, Enschede, The Netherlands, 2005.

[3] S. Van Hoecke, K. Taveirne, K. De Proft, F. De Turck, J. Decruyenaere, B. Dhoedt, An Agent-based Platform for Efficient Telemonitoring Data Processing, Invited Paper/Talk, Health Pervasive Systems Workshop (HPS'06), Lyon, France, 2006.

[4] J. McAffer, J.M. Lemieux, Rich Client Platform Tutorial, EclipseCon, California, 2006.

[5] W. Melhem, D. Glozic, PDE Does Plug-ins, IBM Canada Ltd, http://www.eclipse.org/articles/Article-PDE-does-plugins/PDE-intro.html, 2003.

[6] SAML, OASIS, http://www.oasis-open.org.

[7] WS-Trust, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-trust/ws-trust.pdf.

[8] SAAJ, http://java.sun.com/webservices/saaj

[9] XWS-Security, http://java.sun.com/webservices/docs/1.5/tutorial/doc/XWS-Security.html

[10] XWWS, https://xwss.dev.java.net/

[11] The Java WSIT Tutorial, http://java.sun.com/webservices/interop/reference/tutorial/doc/index.html

[12] Shibboleth, http://shibboleth.internet2.edu/

[13] XACML, http://www.oasis-open.org/committees/xacml.html

PROCEEDINGS OF
THE 2007 INTERNATIONAL CONFERENCE ON
SOFTWARE ENGINEERING RESEARCH & PRACTICE

SERP 2007

Volume II

Editors

Hamid R. Arabnia
Hassan Reza

Associate Editors

Lawrence Chung, Jose Luis Garrido
Emanuel Grant, Vince Schmidt
Ashu M. G. Solo
Nary Subramanian

WORLDCOMP'07
June 25-28, 2007
Las Vegas Nevada, USA
www.world-academy-of-science.org

ISBN: 1-60132-033-7, 1-60132-034-5 (1-60132-035-3)
Printed in the United States of America