Conformance Testing of Open Interfaces in Healthcare Applications - Case Context Management

Tanja Toroi[1][1], Juha Mykkänen[2], Anne Eerola[1]

[1] University of Kuopio, Department of Computer Science, P.O.B 1627, FIN-70211 Kuopio, +358-17-162563, Fax +358-17-16 2595
{tanja.toroi, anne.eerola}@cs.uku.fi
[2] University of Kuopio, Information Technology Centre, HIS Research and Development Unit, P.O.B 1627, FIN-70211 Kuopio
juha.mykkanen@uku.fi

Abstract. In this paper we describe the conformance testing model of the open interfaces developed and applied in the PlugIT project in Finland during 2003-2004. Conformance testing is needed to integrate different software products without a vast amount of extra adaptation work, and to improve software interoperability. The model has been developed and evaluated with co-operation of several healthcare software companies and hospital districts. The clinical context management interface specification is used as an example in this paper.

1 Introduction

The number of information systems and integration needs between these systems is large in healthcare organizations. For example, in Kuopio University Hospital there are more than 180 information systems which have to communicate with each other. At the moment, new systems are integrated to existing ones by tailoring them separately. This is extremely expensive in the long run. If systems have open, standard-based interfaces their interoperability improves, introduction and integration become easier and less local adaptation work is needed. Also, conformance testing is needed to examine if the systems really conform to the standards or other specifications. When certificates (brands) are issued to specification-based software, software developers can use them in marketing and documentation. Also, customers, such as healthcare organizations can benefit from the specifications and brands. They can append specifications to the invitations for tenders and require specification-based software. Software components, which conform to specification can be easily integrated with or replaced by other components.

ISO/IEC defines that conformance is the fulfilment of a product, process or service of specified requirements [8]. A conformance clause is defined as a section of the specification that states all the requirements or criteria that must be satisfied to claim conformance. Conformance testing is a way to verify implementations of the specification to determine whether or not deviations from the specifications exist [10]. Con-

[1] Responsible for correspondence

formance testing is necessary, but not sufficient, for interoperability, because conformance clauses (or specifications) typically only cover some aspects of interoperability.

In conformance testing software products are black boxes; only the interfaces and their relationship to the specifications are examined. In other words, when the software, which offers functionality or services through an interface receives an input, it is tested if the interface can handle and respond to the input correctly.

Conformance testing has been studied quite a lot but there is no proper interface-based testing service available. Australian Healthcare Messaging Laboratory (AHML) [1] has a conformance testing service for the testing of single healthcare messages. The Software Diagnostics and Conformance Testing Division (SDCT), part of NIST's (National Institute of Standards and Technology) Information Technology Laboratory [12] develops software testing tools and methods that improve software quality and conformance to standards. They have made efforts mainly to the XML-based messages and their conformance. Health Level 7 conformance SIG (Special Interest Group) [5] provides a mechanism to specify conformance for HL7 messages and provide a framework for facilitating interoperability using the HL7 standard.

Even if, conformance testing has been studied quite a lot around the world, it has not been performed very much in Finland. In the PlugIT project (see http://www.plugit.fi/english/index.html) we noticed that conformance testing of open interfaces in the healthcare software products need to be examined and elaborated. Thus, we have developed a conformance testing model to test open interfaces. The model consists of four phases: an initial phase, testing performed by the developer, testing performed by the testing lab, and certificate issuing (see Section 4). In our model a HL7 standard, CCOW, is adapted to and only the most essential parts of it have been included to the minimum level specification (see Section 2.2). We have also developed reference implementations to test and evaluate the test cases for conformance testing. Although, we have studied healthcare information systems and their integration similar integration needs can be found in, and our model is applicable to the other domains or integration models.

However, it should be noticed that conformance testing can not be used as verification. Conformance testing only increases the probability that applications are implemented according to the interface specification. Normal software inspections and testing processes must be performed by the developer before conformance testing.

The rest of the paper is organized as follows: In Section 2 the background of the integration and specifications are described. Related work in conformance testing is referred in Section 3. Our conformance testing model is described in Section 4. Discussion and challenges in conformance testing are discussed in Section 5.

2 Background

2.1 Integration Level in Finland

National Project to Secure the Future of Healthcare in Finland has given recommendation in spring 2002: ”The interfaces between healthcare systems shall be made

obliging to all healthcare actors by degree by the Ministry of Social Affairs and Health by the year 2007." The PlugIT project was one step forward by contributing to the implementation of the recommendation by developing solutions for common services and clinical context management, which can be nationally standardized. PlugIT (2001-2004) was a research project, which aimed at decreasing the introduction threshold of healthcare software applications by developing efficient and open standard solutions for integrating them in practice. The results are promising [9]. We have noticed that conformance testing is needed to assure that the applications follow interface specifications so that the application integration and introduction can be done without any extra adaptation or development work. We have developed a model for testing the open interfaces. In the model the applications are given certificates (or brands) if they are in accordance with the interface specification. Specifications developed in the PlugIT project are discussed in Section 2.2 and our model in Section 4.

2.2 Interface Specifications

New means for interoperability have been developed in the PlugIT project by developing healthcare interface specifications for clinical context management (Context manager and Context data interfaces) and common services (User and access rights interface, Patient data interface, and Terminology interface). The developed context management interface specification (see Table 1) is a simplified specification of CCOW (Clinical Context Object Workgroup) standard. CCOW is a vendor-independent standard developed by the HL7 organization to allow clinical applications to share information at the point of patient care [3]. CCOW allows information in separate healthcare applications to be synchronized so that each individual application is referring to the same patient, encounter or user. However, CCOW is too extensive and complex to use in context management in healthcare information systems in Finland. That is why, only the most essential and worthwhile parts of the CCOW standard, i.e. user and patient context management, are included in the minimum level specification.

Table 1. Minimum level context management interface specification

Interface	**Method**	**Raises exceptions**
ContextManager	JoinCommonContext(string applicationName)	AlreadyJoined, TooManyParticipants, GeneralFailure, NotImplemented
	LeaveCommonContext (long participantCoupon)	UnknownParticipant, GeneralFailure, NotImplemented
ContextData	SetItemValues(long participantCoupon, string[] itemNames, string[] itemValues)	UnknownParticipant, NameValueCountMismatch, BadItemNameFormat, BadItemType, BadItemValue, GeneralFailure, NotImplemented

	GetItemValues(long participantCoupon, string[] itemNames)	UnknownParticipant, BadItemNameFormat, UnknownItemName, GeneralFailure, NotImplemented

During the PlugIT project, reference implementations have been implemented and two companies have released commercial implementations of the context management solution. HL7 Finland Association has accepted minimum level context management interface specification as a national recommendation. The specification will be further developed in the SerAPI project [11] and HL7 Finland Common Services SIG (Special Interest Group).

At the moment conformance testing has been performed against the minimum level context management interface specifications, but in future also the other interfaces (e.g., common service interfaces produced by the PlugIT project) will be covered. Examples in this paper deal with context management interfaces.

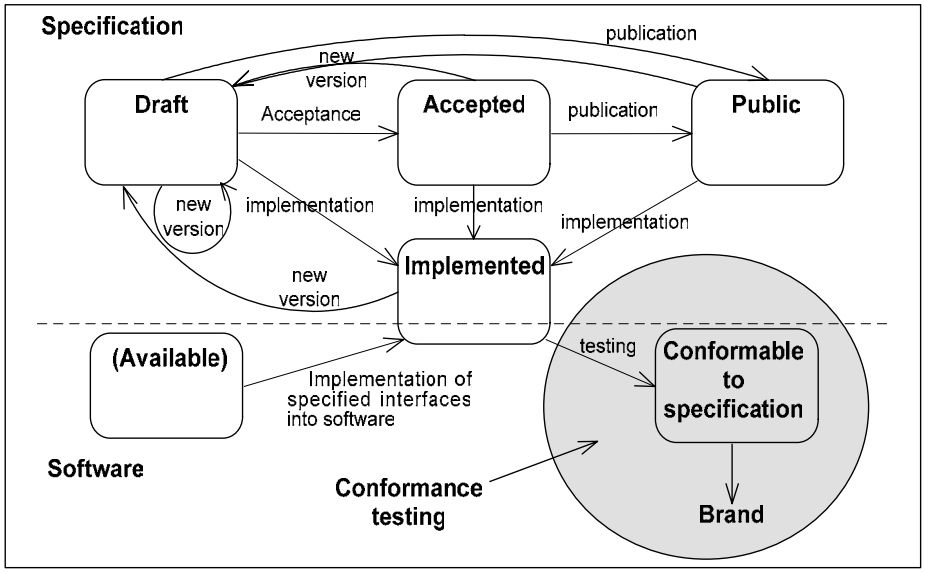

Fig. 1. Relationship between software (below) and specification (above), and conformance testing model in the PlugIT project

The relationship between the software and specifications, and the conformance testing model in the PlugIT project is illustrated in Fig. 1. In the figure, specification is above the dashed line and software is below. Specifications are in different states. At first, specifications are drafts. When a draft specification is accepted by management group it becomes an accepted version of the specification. If an accepted specification is published it becomes a publicly available specification. Software has three states, available, implemented or conformable to specification. An available state

means that there exists the software but it does not necessarily have any implemented interfaces accordant to the specifications. An implemented state means that the specified interfaces have been implemented to software. When the developer has implemented interface implementations to the software, testing has to be performed to verify conformance to the specification. A Certificate issuer / Testing lab performs conformance testing and if the software passes the tests the software becomes conformable to specification state and the brand can be issued to the software. The model is described in more detail in Section 4.

3 Related Work

Australian Healthcare Messaging Laboratory (AHML) [1] is part of the Collaborative Centre for eHealth (CCeH) at the University of Ballarat. AHML provides message testing, compliance and certification services for developers and vendors, system integrators and users within the healthcare industry. Clients can send electronic healthcare messages via Internet to the AHML Message Testing Engine. Messages are tested against healthcare messaging standards. Test cases are built using messaging standards and client-specific requirements, if any. Test history is saved and test reports can be viewed in detail or in summary level.

The Software Diagnostics and Conformance Testing Division (SDCT), part of NIST's (National Institute of Standards and Technology) Information Technology Laboratory is working with industry by providing guidance on conformance topics, helping to develop testable specifications, and developing conformance tests, methods and tools that improve software quality and conformance to standards [12]. SDCT has especially focused on developing XML conformance test suites, and has also written guidance on how to write better specifications.

Health Level 7 (HL7) Conformance SIG (Special Interest Group) attempts to improve interoperability and certification processes [5]. HL7 provides a mechanism to specify conformance for HL7 Version 2.X and HL7 Version 3 messages and provide a framework for facilitating interoperability using the HL7 standard. The problem with HL7 standards is that they are too complex to use in practice and the users are unable to ascertain compliance with standard against their specific needs. As a solution HL7 has developed message profiles, which add specificity to existing messages and identifies scenarios by providing a template for documenting particular uses of HL7 messages. In our model a HL7 standard, CCOW, is adapted to and only the most essential parts of it have been included to the minimum level specification.

Applications can use several interaction styles in application integration, such as direct database access, database-based interoperability adapters, interoperability tables or API-based interfaces with broker techniques [6]. Furthermore, for example, in a layered architecture integration can be performed in workspace, business logic or resource tiers, and every tier has its own special features. We use API-based, workspace tier interface integration in our conformance testing model.

4 Conformance Testing in the PlugIT project

4.1 General

We have developed and used a conformance testing model in the PlugIT project during 2003-2004. Carnahan et al. [2] have introduced that interaction among roles and activities in conformance testing process can be illustrated by a diamond where each role (seller, certificate issuer, testing lab and control board) is in the corner of the diamond. In the PlugIT project certificate issuer and testing lab were not separate roles. Thus, we have adapted the idea and describe the interaction by a triangle. Interaction among roles and activities in conformance testing model in the PlugIT project is illustrated in Fig. 2 (adapted from [2]). A buyer (customer) requires from a seller (developer) that a product is conformable to the specification, and asks the developer to append the brand to the invitation for tenders. The developer develops an application with certain interface implementations and applies for the brand to the application. The testing lab/certificate issuer (=PlugIT project) performs interface testing and issues the brand if the application successfully completes the conformance testing. The control board is answering queries and disputes related to the testing process. The developer is responsible for the product even if it had got the brand and passed the conformance testing.

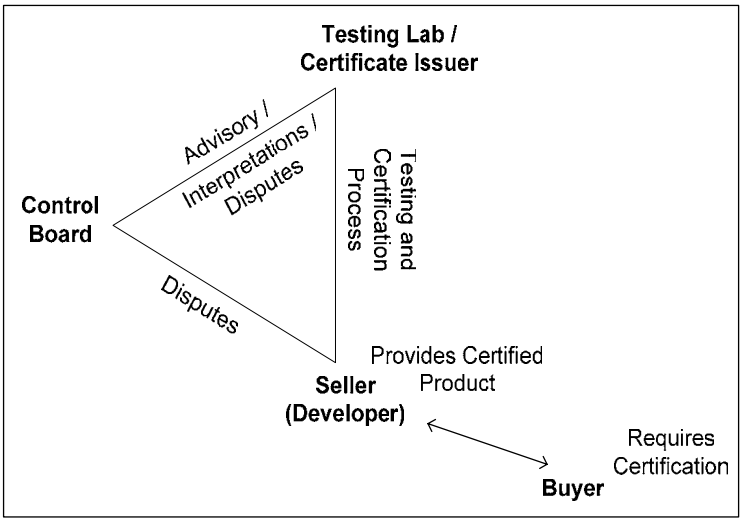

Fig. 2. Interaction among roles and activities (adapted from [2])

The brand is issued to the product which conforms to the specification. To have the brand means that the product has passed the conformance testing process of the open interface specifications. The criterion for issuing the brand is that all the test cases (100%) for required features have to be passed. The brand also means that the product is supported by adequate documentation describing how to introduce, configure and

use the integration solution in the product. The brands have been issued by the PlugIT project. Conformance testing does not remove responsibility from the developer. The developer has to perform normal system and integration testing and documentation before applying for the brand. Conformance testing assures that the interface operates in a tested environment with tested test cases. However, it can not guarantee operation in different environments, as the platform (or infrastructure) of the software is not covered by the integration specification. Developers can use brands in documentation and marketing. Customers can require brands in invitations for tenders.

The brand includes the following information:

- the software product and the version of the software product,
- the interface specification and the version of the specification,
- the level of conformance, if the specification has different conformance levels (e.g. minimum, basic, advanced), and
- date and signatures.

4.2 Phases of the Model

The conformance testing model consists of four phases: An initial phase, testing performed by developer, testing performed by a testing lab, and issuing the brand. The phases will be further discussed in the following subsections.

4.2.1 Initial Phase

In the initial phase, the developer informs the testing lab/certificate issuer that the integration solution is implemented into the product. The developer has to define to which product version the brand is being applied, which interface specifications and what level of conformance is implemented, and who is the contact person for the brand issuing process. The developer delivers the product interoperability description of the implementation to the certificate issuer. The product interoperability description includes all the things concerning interoperability, integration, and introduction, but it does not include any inner details of the implementation. However, implementation-specific additions, expansions, deviations and removals of the specification, if any, must be stated clearly, so not causing contradictions when integrating the implementation with other software. The description includes also instructions for the introduction, installation and configuration. Additionally, it is recommended that the description has usage examples of the integration implementation.

4.2.2 Testing Performed by the Developer

The developer can start this phase when normal integration and system testing, and inspections have been performed and passed. The aim of the conformance testing performed by the developer is to assure conformance and to fix failures before applying for the brand. Developer tests the product with test cases received from the testing lab and with own test cases. It is essential that only the interfaces in the specification are tested, not the whole implementation and implementation-specific features. In the PlugIT project reference implementation of the context management server was developed to support the testing of the client part. Test cases (see Section 4.3) were de-

veloped using the black box testing method and the reference implementation. Test cases were sent to the developer for the self testing.

4.2.3 Testing Performed by the Testing Lab/Certificate Issuer

The aim of the testing in the testing lab is to complement testing, to ensure conformance to the specification, and to make sure that the implementation is functioning in practice. PlugIT project itself acted as both testing lab and certificate issuer. Testing was organized in practice using reference implementations. Testing environments were both in developer's development environment and customer's introduction environment.

Not all the same test cases as in the previous phase are necessarily retested. Which of the test cases have to be retested depends on resources and situation, such as reliability of the developer, and how critical the interface is. We need to study this further to get as reliable and adequate test cases as possible. It is also possible to test the product together with the other products which have already gained the brand. If the testing is passed, a test report, which includes the summary of the test cases and their pass/fail information, is generated (see Section 4.4). Also the documentation is checked if it contains necessary information for the introduction of the integration solution.

4.2.4 Issuing the Brand

The last phase in the conformance testing model is the brand issuing. If all the tests have been passed and the documentation is adequate the certificate issuer issues the brand and delivers it to the contact person of the developer. Test reports are made public and published, for example, on the web page, with the permission of the developer.

4.3 Test Cases

In this Section we give some examples of the context management test cases used in conformance testing in the PlugIT project. The basic context management operation flow follows, for example, the following path:

- A client application joins the context management (JoinCommonContext).
- The client application asks a user context (GetItemValues).
- The client application asks a patient context (GetItemValues).
- The client application changes the patient (SetItemValues).
- The client application updates the patient context (SetItemValues).
- The client application leaves the context (LeaveCommonContext).

Test cases are generated based on the previous path and on error situations. Each test case includes definition, precondition, input, output, and pass/fail information. Input is in URL format and is suitable for testing of the interfaces. Some of the values must be replaced with site-specific or application-specific parameters, such as IP address and application names. Some test cases can have an input value that is an output value of another operation, such as participantCoupon is an output value of JoinCommonContext and an input value for SetItemValues. Examples of the test cases in

conformance testing are described in Tables 2-4. First there are two general context management operations: joining the context and getting the values from the context. Next there is a typical failure where the coupon, which is used to the identification of the session, is used after leaving the context.

A test input (see Table 2) and its different parts are built up as follows:

http://url.fi/cm?interface=ContextManager&method=JoinCommonContext&param=...

Network address of the called component — Interface — Method — Parameters

In the previous test input the Context Manager (cm) component is located in the network address http://url.fi/cm and its interface ContextManager and method Join-CommomContext are called with certain parameters.

Table 2. JoinCommonContext

Definition:	Client application joins the context management.
Precondition:	Application name must be accepted and another application has not joined the context with the same application name.
Input:	http://193.167.225.119/cm.pp?interface=ContextManager&method=JoinCommonContext&applicationName=LoginMaster
Output:	participantCoupon=11900200
Pass/Fail:	

Table 3. GetItemValues

Definition:	Client application asks the patient context
Precondition:	Application has joined the context, item has been set, participantCoupon has been received when joined.
Input:	http://193.167.225.119/cm.pp?interface=ContextData&method=GetItemValues&participantCoupon=11900347&itemNames=Patient.Id.NationalIdNumber
Output:	itemValues=Patient.Id.NationalIdNumber|220345-XXXX
Pass/Fail:	

Table 4. Coupon used after leaving the context

Definition:	Coupon used after leaving the context.
Precondition:	Application has left the context, participantCoupon has been expired.
Input:	http://193.167.225.119/cm.pp?interface=ContextData&method=GetItemValues&participantCoupon=11900347&itemNames=Patient.Id.NationalIdNumber
Output:	e.g. exception=GeneralFailure&exceptionMessage=General failure
Pass/Fail:	

4.4 Test Report

If conformance testing is passed the test report, which includes the information about the implementation under test, implementation-specific settings and special considerations, and test cases with the pass/fail information, is published. The information of the implementation under test describes the product and its version, the interface specification and its version, the role of the application in the integration, names of the testers, and date. Application-specific settings and special considerations could be, for example, accepted application and item names (optional features in context servers), address of the service, number of joining applications, and installation instructions. Additionally, any special information, which must be taken into consideration in integration is informed. Test cases are reported similarly as in Section 4.3. Inputs and outputs are written to the log file but they are not in the public test report.

4.5 Present State of the Model

The conformance testing model has been developed and evaluated with co-operation of several healthcare software companies and hospital districts. The evaluation has been performed for a context management server implementation. At the moment, there are several commercial implementations of the context manager, of which one has received a brand for the context management implementation. Conformance testing revealed different interpretations of the context management specification. They were corrected to the product in question, and clarified in further versions of the specification. At the moment, we are studying the conformance testing processes more and our model is further elaborated.

5 Discussion and Challenges in Conformance Testing

The conformance testing model developed in the PlugIT project is on one hand a light and rapid model and on the other hand it assures that products conform to the requirements of the specifications. However, the model is not complete. Some challenges, which have to be solved before applying the model further, are described here.

Software product versions are introduced in rapid cycles. How to do the re-evaluation of the product? Rosenthal et al. [10] have presented that a brand could be issued for a longer period (e.g. 2 years) if no errors are found in conformance testing. In our case, new software versions can be released, for example, once every two week. Thus the brand has to be renewed much more often and re-evaluation has to perform automatically using, for example, web-based testing services.

The test data can not be real patient (or production) data. We have to create test cases which correspond as much as possible to the real (patient) scenarios. However, to be able to create such scenarios we need to have good domain knowledge. In our case, the testing model has to be flexible enough to cover both testing the client and server parts of context management solutions, and also other types of service interfaces, such as common services. The conformance testing model has to also support testing of the workflow. Therefore, test cases have to constitute a flow of test cases, in

which the order of the execution matters. All these different types of solutions require different test cases.

Some requirements of the solution (e.g. context server) are implicit. For example, the context management interface specification does not clearly state, that setting context from one workstation must not affect contexts set from other workstations. However, it is a basic requirement for server-based applications, which contain workstation-specific data, such as context management. Thus, integration specifications must contain enough information about the requirements for the solutions, in addition to mere interface signatures.

Some parameters are acquired during the execution chain of the test case. Some of them are static for a given environment, but several are specific to the applications used or to the implementation of the server. A standard way of identifying and classifying this sort of parameters for test case definitions is needed. Furthermore, some requirements for the parameters can not be easily tested. For example, the specification states, that the participantCoupon parameters returned by the service must be unique during the execution of the service. Complete testing for such uniqueness would require far too many different test cases in practice.

Extension points in specifications (such as an optional feature for context servers to accept only some named applications), which are not required features must be tested, if the specification has conformance levels for such extensions. Thus, the integration specifications and standards should be developed to express clearly, which options are implementation-specific or optional. In addition, specifications should provide guidance on how should implementation-specific features be documented and used.

In our context management case, the contents of the parameters in test cases are quite simple and have well-specified semantics. However, when testing complex data structures (e.g. patient records), value sets for the parameters, and interpretations of different operations and data elements must be precise. Versioning of these value sets (e.g. different versions of disease classifications) further complicates the interoperability and conformance testing.

If software customers coherently require standard-based ("branded") interfaces in invitation for tenders, the quality and interoperability of solutions is improved. This does not prevent free competition, but promotes standardization in widely-used interfaces and reduces local “fixes”. However, customers need advice when gathering and setting their requirements. Interface specifications do not currently contain all the needed information, including basic requirements, conformance levels, and different types of parameters, which must be conformed to. Questions of the customers have to be answered and quality of interoperability specifications improved in order to get software customers to demand certified and interoperable software products.

As we can see, interface standards are only one part of interoperability. There can be products that conform to the specifications but are not interoperable. There can also be products that do not exactly conform to the specification but can be interoperable with other products [4]. Thus, when developing interface specifications and conformance testing models all the things influencing integration and interoperability have to be taken into consideration and specified. One step in this direction is integration profiles by IHE initiative (Integrating Healthcare Enterprise) [7]. IHE integration profiles offer a common language that healthcare professionals and software developers can use in communicating requirements for the integration of products. Integration

profiles describe real-world scenarios or specific sets of capabilities of integrated systems.

Our conformance testing model has been successfully used in conformance testing of healthcare application interfaces. Although the model is simple and the test cases as well as the specification are inadequate for comprehensive interoperability, the model is a good starting point to develop more efficient model for conformance testing. The model will be further elaborated in our next projects, OpenTE and SerAPI. In the OpenTE project testing environments for several web-based testing services, for example, CDA R1, CDA R2, minimum level context management, and common service interfaces will be developed. We have to also resolve how to handle conformance testing for healthcare application integration in national (even international) level in future and who will issue the brands.

6 Acknowledgement

This paper is based on research in the PlugIT project (2001-2004), funded by the National Technology Agency TEKES, software companies and hospital districts. The authors thank all the involved participants in these projects.

References

1. Australian Healthcare Messaging Laboratory. Referred October 12 2004. URL: http://www.ahml.com.au/
2. Carnahan L., Rosenthal L., Skall M.: Conformance Testing and Certification Model for Software Specification. ISACC '98 Conference (1998).
3. CCOW information for the healthcare industry. Referred October 12 2004. URL: http://www.ccow-info.com/
4. Eichelberg M., Riesmeier J., Jensch P.: DeNIA: Computer Supported Interoperability Assessment for DICOM Devices. In: Niinimäki J., Ilkko E., Reponen J., eds. EuroPACS 2002. Proceedings of the 20th EuroPACS annual meeting, Oulu university press (2002) 55-58
5. Health level 7 (HL7). Referred November 5 2004. http://www.hl7.org/
6. Herzum P., Sims O.: Business Component Factory. Wiley Computer Publishing, New York (2000)
7. HIMSS, RSNA: Integrating the Healthcare Enterprise - IHE Technical Framework Volume I - Integration Profiles, Revision 5.3. HIMSS/RSNA. (2002)
8. ISO/IEC Guide 2: 1996 Standardization and Related Activities: General Vocabulary (1996)
9. Mykkänen J, Porrasmaa J, Rannanheimo J, Korpela M.: A process for specifying integration for multi- tier applications in healthcare. Int. J. Med. Inf. 70(2-3) (2003) 173-182
10. Rosenthal L., Skall M., Carnahan L.: White paper. Conformance Testing and Certification Framework. NIST (2001)
11. Service-oriented Architecture and Web Services in Healthcare Application Production and Integration. Home page of the SerAPI project. Referred Oct 25 2004. URL: http://www.uku.fi/tike/his/serapi/english.html
12. The Software Diagnostics and Conformance Testing Division, Information technology Laboratory, National Institute of Standards and Technology. Referred Oct 12 2004. URL: http://www.itl.nist.gov/div897/