Semantic Annotation, Publication, and Discovery of Java Software Components: An Integrated Approach

Zinon Zygkostiotis[1], Dimitris Dranidis[1, 2], Dimitrios Kourtesis[2]

[1] *Computer Science Department, CITY College,*
Affiliated Institution of the University of Sheffield,
Tsimiski 13, 54624 Thessaloniki, Greece
zzygkostiotis@city.academic.gr, dranidis@city.academic.gr

[2] *South East European Research Centre (SEERC),*
Research Centre of the University of Sheffield and CITY College
Mitropoleos 17, 54624, Thessaloniki, Greece
dkourtesis@seerc.org

Abstract: Component-based software development has matured into standard practice in software engineering. Among the advantages of reusing software modules are lower costs, faster development, more manageable code, increased productivity, and improved software quality. As the number of available software components has grown, so has the need for effective component search and retrieval. Traditional search approaches, such as keyword matching, have proved ineffective when applied to software components. Applying a semantically-enhanced approach to component classification, publication, and discovery can greatly increase the efficiency of searching and retrieving software components. This has been already applied in the context of Web technologies, and Web services in particular, in the frame of Semantic Web Services research. This paper examines the similarities between software components and Web services and adapts an existing Semantic Web Service publication and discovery solution into a software component annotation and discovery tool which is implemented as an Eclipse plug-in.

1. Introduction

The advent of rapid application development has led to an ever increasing emphasis on software reuse. Component-Based Software Development (CBSD) emphasises the reuse of existing code from either in-house repositories or 3rd party vendors, and has been shown to result in lower development costs, faster time-to-market, more effective maintenance and application upgrade, increased programmer productivity, and improved overall software quality [4, 18].

With software components being stored in code repositories, private or public ones, these repositories can potentially become extremely large. As they grow in

number and in size, so does the need to be able to search them effectively and retrieve component information and specifications. To enable this, there needs to be a standard way of representing this component-related information, thus facilitating Computer-Aided Software Engineering (CASE) tools in the discovery and retrieval of relevant results.

A number of search solutions have been proposed, developed and implemented for this purpose to date [20, 19], ranging from basic keyword searches to more advanced methods such as signature and behaviour matching using formal logic-based techniques. Traditional search approaches, such as keyword matching, are effective when searching Web pages and text documents. However, they have proven to be very inefficient when applied to software components. One of the reasons for this is that it is extremely difficult to convey sufficiently expressive domain-related information through a component's name or description.

The use of Semantic Web technologies in the annotation of software component information has enormous potential in achieving better targeted searches and more meaningful and accurate search results [20, 19, 14, 2]. Adding machine-processable semantic information to components and publishing this information in a standard way would make the classification, search and retrieval of components more effective, and would thus enable greater utilisation and easier integration of the vast number of software components currently available.

This paper presents an integrated approach for the annotation, publication and discovery of reusable Java software components through the use of Semantic Web technologies. We propose a method for annotating Java source code using domain ontologies that have been encoded in OWL-DL [10], and a means to publish and subsequently discover the resulting semantic descriptions using a semantic registry which employs Description Logic (DL) reasoning to perform matchmaking among software component advertisements and requests. Our approach has been validated through the development of a fully functional plug-in for the Eclipse IDE that supports all three facets of the approach: Java code annotation, publication of semantic descriptions, and search of software components. Our approach and implementation builds on earlier research work in the area of semantically-enhanced publication and discovery of Web Services, and relies on an existing open source semantic service registry for publication and discovery.

This paper is organised as follows. In Section 2 we look at various approaches for description and discovery of software components, explore similarities between software components and Web services, and report on a recently developed approach for publication and discovery of Web services with a semantically-enhanced service registry. Section 3 describes how this system can be adapted for use with software components and details an Eclipse plug-in developed for this purpose. We also provide an overview of the semantic matchmaking process when searching for software components, and outline the benefits this approach can bring over the use of traditional keyword-based and signature-based retrieval methods. Section 4 concludes the paper with a summary of the main points in this work.

2. Background of the Approach and Related Work

The basis of Component-Based Software Engineering (CBSE), also referred to as Component-Based Software Development (CBSD), is that certain functions and parts of large software systems appear numerous times within the system; therefore they should only be written once and not repeatedly throughout the application. Encompassing the required functionality into pluggable components and defining interfaces independent of any domain details allows components to be reused.

2.1 Software Component Retrieval Approaches

When the idea of software componentisation was first proposed by McIlroy in 1968 [11], it was recognized that a key requirement would be the indexing and retrieval of components. Currently, there is no universal agreement as to what information is required to describe software components such that they can be effectively retrieved. Existing code repository implementations tend to use proprietary or non-standard syntax and semantics for component descriptions and indexing, often employing quite elaborate classification schemes [3, 9, 8]. This inevitably makes searching in different code repositories even more difficult.

Mili et al. [12] group the types of search used for software component retrieval into four basic categories: simple keyword-based text searches, faceted classification and retrieval, signature matching and behaviour matching. Other research has classified all or some of these types into similar categories, such as Ostertag et al. [15], who also describe methods for free-text keyword searching and faceted index searching.

Keyword-based searching is the simplest approach to implement and is the one that the majority of search engines use. The successful retrieval of relevant components using this method is highly dependent on the original names given to the components and cannot take into account such information as relationships between components, their execution context and synonymous keywords. A software component retrieval scheme based on this approach is described in [13].

Faceted classification and retrieval involves extracting keywords from component descriptions and documentation and arranging this information into a predefined classification scheme or taxonomy. Although such an approach has been shown to be quite effective in the retrieval of relevant search results [15], it is only effective if the components fit into the classification scheme being used. Hence, a significant effort is required to maintain such classification schemes.

The signature matching approach, such as that described in [9], is rather detached from the application domain in that it attempts to describe components based on input and output parameters, creating a signature based on a mathematical algorithm. However, components having matching signatures are not guaran-

teed to be related. Behavioural matching extends signature matching somewhat in that it attempts to also describe the particular behaviour of a component. According to Suguraman et al. [19], both these approaches are cumbersome and inefficient.

2.2 Software Components vs. Services, and Semantic Retrieval

The similarities and differences of software components and Web services is regularly discussed throughout much of the literature in the field of CBSE. In [1], Breivold and Larsson provide a comparison framework for component-based and service-oriented software engineering and discuss the research efforts that have been done in combining the strengths of the two.

Yao et al. [20] suggest that there is little difference between software components and Web services, going as far as saying that a reusable component is in fact a service, and on this basis, the description and matching technologies employed for Semantic Web Services are just as applicable to software component description, classification and retrieval. Korthaus et al. [6] also investigate the use of Semantic Web technologies in the field of CBSE, and argue that CBSE can greatly benefit from the use of existing Semantic Web technologies for component classification, publication and discovery.

Paar [16] describes a Microsoft Visual Studio add-in developed for annotating C# source code with semantic information using ontologies encoded in DAML (the precursor of the OWL Web Ontology Language which is now a W3C standard) and WSDL (Web Service Description Language). The system annotates C# source code with references to ontology concepts and then extracts this information, converting it into a specially-adapted and semantically-extended form of WSDL. This WSDL file can then be used to advertise the component in much the same way as one would do for Web services.

Similarly, Yao et al. [20] describe a semantics-based approach to component classification and retrieval where software components are annotated with DAML ontologies. The component annotations and user queries are described in a WSDL format and then translated into "conceptual graphs", which are then used in their semantic matchmaking process. They also employ a software component repository based on the UDDI standard. However, one of the limitations they discuss was the lack of semantic support in both WSDL and UDDI.

2.3 The FUSION Semantic Registry

The use of Semantic Web technologies to represent Web service properties and the introduction of semantic matchmaking functionality in service registries (primarily UDDI) has been the focus of several works in recent years, generally within the field of Semantic Web Services (SWS) research. Kourtesis et al. [7]

present an approach that is focused at automating the evaluation of Web service integrability on the basis of the input and output messages that are defined in the service's interface. The approach has been applied during the development of the FUSION Semantic Registry, a semantically-enhanced service registry utilised in research project FUSION and released as open source software.

The aim in integrability-oriented service matchmaking within the FUSION Semantic Registry is to detect if interoperability at the level of data can be guaranteed among an advertised service and its prospective consumer, such that proper data flow and communication can take place. In plain terms, we seek to ensure that the data that the consumer is able to provide upon invocation are sufficient with regard to the input data that the advertised service expects to receive, and conversely, the output data that the advertised service produces are sufficient with regard to the data that the consumer expects to receive. This relates directly to the notions of covariance and contravariance applied in the context of function subtyping and safe substitution, which have been studied in detail within type-theory and object-oriented programming research [17].

In order to represent the functional and non-functional service properties that are of interest for matchmaking one needs to create a *Functional Profile* and define its key attributes in terms of references to an ontology encoded in OWL-DL. A *Functional Profile* is expressed as an OWL class with three types of object properties: *hasCategory* (representing the service's functional categorisation), *hasInput* (representing the service's set of input data parameters) and *hasOutput* (representing the service's set of output data parameters). There must always be one category declared, and zero or more inputs and outputs.

The purpose of describing services as OWL-based functional profiles is to enable semantic matchmaking among service advertisements and requests. When a service provider publishes a service advertisement, the service's profile is stored in the registry as an *Advertisement Functional Profile (AFP).* During the discovery process the requestor constructs a profile describing the desired service, i.e. a *Request Functional Profile (RFP).* Matchmaking among the two is performed through subsumption checking with a Description Logics reasoner (Pellet). Details on the publication and discovery algorithms are provided in [7].

3. Semantic Annotation, Publication and Discovery

In this paper we propose to build on the FUSION Semantic Registry infrastructure for classification, publication and retrieval of reusable software components. The following subsections detail how this can be realised in the form of an Eclipse plug-in for Java source code. Section 3.1 describes how components are described through ontology-based annotations. Section 3.2 provides an overview of the Eclipse plug-in and the functionality it offers. Finally, section 3.3 looks at the way in semantic matchmaking process is carried out within the Semantic Registry.

3.1 Semantic Annotations for Java (SA-Java)

In order to adapt the publication and discovery mechanisms of the FUSION Semantic Registry to cater for software components, we need a method of describing components similar to the one used for describing services, i.e. we need software component functional profiles. A Software component profile contains all information required to semantically describe, publish and search for reusable Java code in our approach. The information it holds is outlined in Table 1.

Table 1. Attributes of Advertisement Software Component Profiles

Attribute	Description
Identifier	A name given to the component for readability. It plays no part in semantic publication or discovery process but allows the component to be found via keyword-based search.
Description	A free-text description of the component. As with the identifier, it plays no part in the semantic publication or discovery process.
Category	The URI of an OWL concept describing the category to which the user has chosen to classify the component.
Inputs	The URIs of one or more OWL concepts describing the inputs the component expects. If this field is empty, the component does not require inputs.
Outputs	The URIs of one or more OWL concepts describing the outputs the component returns. If this field is empty, the component does not return any values.
RepositoryURI	The location where the component or component source code can be found. This can be a local or network file system location, Web URL or CVS/SVN repository location.

Advertisement Software Component Profiles are constructed automatically by the registry at the time of publication. Part of the information that is required for constructing them (Identifier, Description, and RepositoryURI) is obtained from the user, while the rest (Category, Inputs, and Outputs) is obtained by parsing the source code and retrieving semantic annotations placed there by the developer.

The method that we employ for source code annotation makes use of the standard annotation facility that was introduced by Sun with the release of Java 5.0 [5]. This allows adding metadata to code elements such as package declarations, type declarations, constructors, methods, fields, parameters, and variable declarations. The Java 5.0 platform comes with some predefined annotation types, but also allows developers to define their own types.

For the needs of our approach we have defined three types of annotations. The *Category* annotation is used to classify a Java class or method with regard to an ontological concept describing its purpose or application domain. For example, if a class contained methods and functions related to cash transactions from ATM machines, then the category annotation would provide a reference to an OWL concept describing this. Similarly, the *Input* and *Output* annotations are used to classify the inputs and outputs in terms of domain objects. The code snippet below

shows how a Java method annotated in this way could look.

```
@SAJavaCategory("http://www.seerc.org/onto.owl#ATM_Services")
public
@SAJavaOutput("http://www.seerc.org/onto.owl#Loggon_Confirmation")
boolean logon (
      @SAJavaInput("http://www.seerc.org/onto.owl#Card_Number")
      String userId,
      @SAJavaInput("http://www.seerc.org/onto.owl#PIN")
      String password )
{
    // method body
}
```

3.2 SA-Java Eclipse Plug-In

Our approach comes with a tool that interfaces with the FUSION Semantic Registry and supports Java source code annotation, publication of semantic descriptions, and search and retrieval of software components. The tool was developed as a plug-in for the popular Eclipse IDE and offers two separate views: annotation of source code is provided by the *Annotator view*, while component publication and discovery is provided by the *Semantic Registry view*. A screenshot of the SA-Java plug-in in the Eclipse workbench is shown in Figure 1.

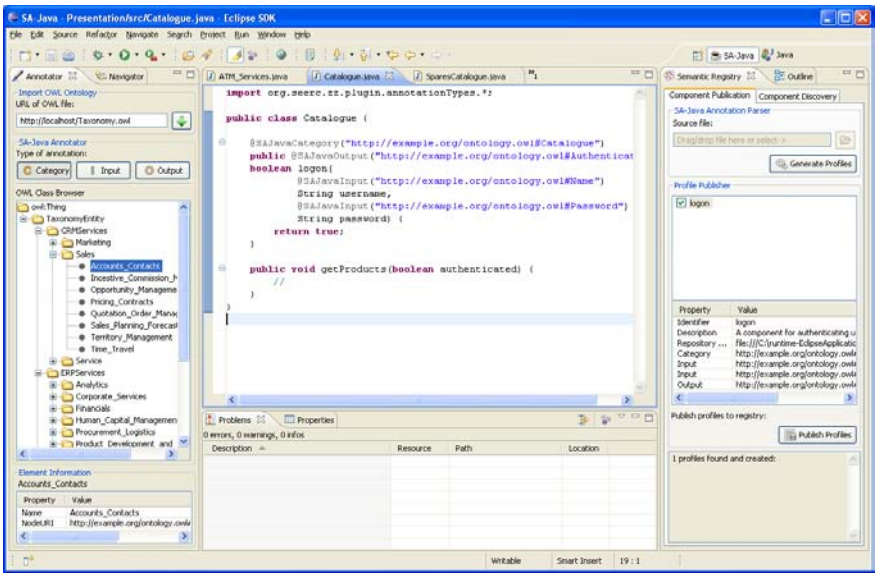

Figure 1. The SA-Java Plug-In and its different views

In order to annotate a Java source file with semantic information, an OWL file is first loaded into a browser in the Annotator view where the classes described in the OWL file can be examined. The user then selects the type of annotation required and, by *dragging and dropping* the OWL class directly from the browser to a point in the source code, the respective annotation is added.

Publication involves parsing an already annotated source file and creating candidate profiles. The plug-in scans a file for annotations and creates candidate profiles based on the annotated information that is found. A wizard is presented to the user who can examine the generated candidate profiles in turn, and edit or add any necessary information. On completion of the wizard, all generated candidate profiles are imported into the Semantic Registry view. The user can then select which of the candidate profiles should be actually published.

Component discovery is accomplished by creating Request Software Component Profiles. It is the profiles themselves that are used as search parameters rather than the traditional keyword text based approach most people are familiar with. The Registry view has a Profile Builder for this purpose where users can create software profiles which can then be sent to the registry for semantic matching. Any component profiles found in the registry that match the one sent as the search parameter are returned.

3.3 Semantic Matchmaking of Components

To illustrate the semantic matching process employed in the FUSION Semantic Registry, we use the examples of three methods whose profiles are detailed below.

Advertised profile of Class A *logon* method:
hasCategory: http://www.seerc.org/onto.owl#**Catalogue**
hasInput: http://www.seerc.org/onto.owl#**Name**
hasInput: http://www.seerc.org/onto.owl#**Password**
hasOutput: http://www.seerc.org/onto.owl#**Authenticated**

Advertised profile of Class B *signin* method:
hasCategory: http://www.seerc.org/onto.owl#SparesCatalogue
hasInput: http://www.seerc.org/onto.owl#Name
hasInput: http://www.seerc.org/onto.owl#Password
hasOutput: http://www.seerc.org/onto.owl#Authenticated

Advertised profile of Class C *logon* method:
hasCategory: http://www.seerc.org/onto.owl#ShoppingCart
hasInput: http://www.seerc.org/onto.owl#Name
hasInput: http://www.seerc.org/onto.owl#Password
hasOutput: http://www.seerc.org/onto.owl#Authenticated

Note the usage of OWL concepts for describing the different parts of the profiles. For instance, the category of the profile for the logon method of Class A has the value of *Catalogue*, which signifies that the profile either models or is related to a Catalogue object in the domain. The other two profiles have been categorized

as *SparesCatalogue* and *ShoppingCart*. For the purpose of this discussion let us assume that there exists an OWL-encoded taxonomy hierarchy in which the *SparesCatalogue* concept is defined as a subclass of *Catalogue*, whereas the *ShoppingCart* concept is a sibling class of *Catalogue.*

One thing we can observe in the example profiles is that all three require the same basic information as arguments and return the same result, regardless of the names they have been given in the method declarations and, perhaps more significantly, of the Java data types used for the arguments. Semantic annotation and profile generation makes no distinction between the Java data types used for elements, only what they represent.

As described earlier, to carry out a search using this approach, a request profile must be created that describes the required component. For example, we might be interested in finding any component that provides a method which accepts a username and a password as arguments and returns a response whether authentication has been successful, as in the following request profile:

Request Profile 1

```
hasCategory: http://www.seerc.org/onto.owl#Thing
hasInput:    http://www.seerc.org/onto.owl#Name
hasInput:    http://www.seerc.org/onto.owl#Password
hasOutput:   http://www.seerc.org/onto.owl#Authenticated
```

Searching using this profile would return all three of the advertised classes. This is because all three require two arguments that represent usernames and passwords (regardless of the Java data types used) and return a response indicating whether authentication has been successful. Also, their categorizations are all subclasses of the *Thing* concept (note that owl:Thing is the top concept in every OWL ontology and by definition subsumes every other possible concept).

We could modify the above profile to search for components that could be modelled as *Catalogue* objects. For this, we would need to replace the *hasCategory* URI with "http://www.seerc.org/onto.owl#Catalogue". This time, only classes A and B would be returned as matching, as they have been categorized as either of type Catalogue, or *SparesCatalogue* (which is a subclass of *Catalogue*). Class C's categorization is unrelated and so it would be excluded from the returned results.

The above example is a simple illustration of the semantic matching process based on the category classification of the profiles. The same procedure is applied when matching other elements of the profile, that is, the inputs and outputs, with even more interesting results. For example, the *Name* concept would also match any concepts that are a subclass of *Name*. When developers construct request profiles, what they are specifying is the number and types of inputs they can provide and the number and types of outputs they expect. In other words, they require a component that is related to a specific category and can return at least the outputs requested given at most the inputs that can be provided. To illustrate this, let us examine the following request profile.

Request Profile 2
hasCategory: http://www.seerc.org/onto.owl#SparesCatalogue
hasInput: http://www.seerc.org/onto.owl#Name
hasInput: http://www.seerc.org/onto.owl#Password
hasInput: http://www.seerc.org/onto.owl#EmployeeId
hasOutput: http://www.seerc.org/onto.owl#Authenticated

Searching using this profile would still return Class B even though the request profile has an extra input. This is because Class B can still provide the required output with only two of the three inputs the requestor is able to provide. Class B can therefore be utilized, and the extra input, *EmployeeId*, could be ignored. The opposite, however, is not true. Take, for example, the following request profile:

Request Profile 3
hasCategory: http://www.seerc.org/onto.owl#SparesCatalogue
hasInput: http://www.seerc.org/onto.owl#Password
hasOutput: http://www.seerc.org/onto.owl#Authenticated

This would return none of the three advertised classes. This is because they all require at least two inputs but the requestor here states that only one can be provided. Therefore, the components would not have enough information with which to carry out their tasks. The matching procedure with the outputs is similar but reversed. In this case, there is a match if the advertised profile can provide at least the outputs required by the requestor – any others can be ignored.

Hence we can see that applying a semantics-based approach to component search and retrieval is far more effective than traditional search approaches. Keyword-based and signature-based matching approaches cannot distinguish between components that display the same name/different functionality or different name/same functionality properties. Applying semantics not only goes a long way in solving this problem, but can also match components that can fulfil a request even if they are not a direct match.

4. Conclusions

The work presented has shown how Semantic Web technologies can be applied to CBSE, in particular, to the annotation, publication and discovery of software components. We proposed a method for annotating Java source code using domain ontologies encoded in OWL-DL, and a means to publish and subsequently discover the resulting semantic descriptions using a semantic registry which employs DL reasoning to perform matchmaking among advertisements and requests. Our approach is supported by a fully functional plug-in for the Eclipse IDE that supports annotation, publication and discovery of components, and is shown to offer significant benefits for retrieval of software components over the use of traditional approaches such as keyword- or signature-based matching.

References

1. Breivold H.P., Larsson M. (2007). Component-Based and Service-Oriented Software Engineering: Key Concepts and Principles. Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, pp.13-20.
2. Dong J.S. (2004). Software Modeling Techniques and the Semantic Web. Proceedings of the 26th International Conference on Software Engineering, pp. 724-725.
3. Graubmann P., Roshchin M. (2006). Semantic Annotation of Software Components. Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 46-53.
4. Haines G., Carney D., Foreman J. (2007). Component-Based Software Development / COTS Integration. Carnegie Mellon Software Engineering Institute, Pittsburgh.
5. JDK 5.0 Documentation. (2004). JDK 5.0 Developer's Guide: Annotations. Sun Microsystems Inc. http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html.
6. Korthaus A., Schwind M., Seedorf S. (2007). Leveraging Semantic Web Technologies for Business Component Specification. Journal of Web Semantics: Science, Services and Agents on the World Wide Web, vol., no. 2, pp. 130-141.
7. Kourtesis D., Paraskakis I. (2008). Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery. Proceedings of the 5th European Semantic Web Conference, LNCS 5021, pp. 614-628.
8. Lee J., Kim J., Shin G. (2003). Facilitating Reuse of Software Components using Repository Technology. Proceedings of the 10th Asia-Pacific Software Engineering Conference, pp. 136-142.
9. Luqi, Guo J. (1999). Toward Automated Retrieval for a Software Component Repository. Proceedings of the 6th Symposium on Engineering of Computer-Based Systems (ECBS '99), pp. 99-105.
10. McGuinness D.L., van Harmelen F. (2004). OWL Web Ontology Language Overview, W3C Recommendation
11. McIlroy D. (1968). Mass-Produced Software Components. Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany, pp. 88-98.
12. Mili R., Mili A., Mittermeir R.T. (1998). A Survey of Software Storage and Retrieval, Annals of Software Engineering, vol. 5, no. 2, pp. 349-414.
13. Mili A., Mittermeir R. (1994). Storing and Retrieving Software Components: a Refinement Based System. Proceedings of the 16th International Conference on Software Engineering (ICSE-16), pp. 91-100.
14. Oberle D., Eberhart A., Staab S., Volz R. (2004). Developing and Managing Software Components in an Ontology-Based Application Server. Proceedings of the 5th International Middleware Conference, LNCS 3321, pp. 459-477.
15. Ostertag E., Hendler J., Prieto-Diaz R., Braun C. (1992). Computing Similarity in a Re-Use Library System - an AI Approach. ACM Transactions on Software Engineering and Methodology, vol. 1, no. 3, pp. 205-228.
16. Paar A. (2003). Semantic Software Engineering Tools. OOPSLA '03, Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pp. 90-91.
17. Simons A.J.H. (2002). The Theory of Classification, Part 4: Object Types and Subtyping. Journal of Object Technology, vol. 1, no. 5. pp. 27-35.
18. Szyperski C. (1998). Component Software: Beyond Object-Oriented Programming: Addison-Wesley.
19. Sugumaran V., Storey, V.C. (2003). A Semantic-Based Approach to Component Retrieval. SIGMIS Database, vol. 34, no. 3, pp. 8-24.
20. Yao H., Etzkorn L. (2004). Towards a Semantic-based Approach for Software Reusable Component Classification and Retrieval. Proceedings of the 42nd Annual Southeast Regional Conference, Huntsville, Alabama, pp. 110-115.