2
Background

In this section we present definitions, important to gain an understanding of what we mean with some terms used in this work.

2.1.
Software Reuse, Software Repositories & Agent Components

Software reuse is the technique of recovering existing software artifacts for software development as a strategy to increase the efficiency of development. This increases productivity and quality of software systems. Software reuse techniques are widely used in other software engineering contexts like object oriented application frameworks and libraries, software architectures and patterns. Component based (CB) software engineering has built upon reuse techniques to form a model for software development and system construction. The foundation of a CB development technology is based on its software component model, which represents components and their composition mechanisms. One of the main characteristics of software components is their use via well-defined interfaces. The interface of a component is the communication method for interconnecting with other components, and it is the interface which provides the structural and functional information that is used to check the degree of similarity and identify and estimate reusable parts of a component. Identifying a reusable component and its functionality is important to increase productivity of CB development. Successful reuse requires a wide variety of high quality components, proper classification and effective retrieval mechanisms. How reuse occurs more frequently with well-known components, a search engine must help users in exploring the source and gaining knowledge about other components that are similar to the initial target, facilitating future reuse.

Component repositories are required to facilitate efficiently the reuse of dissimilar varieties of software components; e.g., design diagrams, requirements, specifications, documentation, test-cases; providing functionalities for component

reuse process such as specification, storing, organizing, querying and presenting in such way that the access is easy, reducing browsing time and increases productivity for users. To be collectively managed, the previously purchased and generated components should be stored and can be shown to let users know the list of reusable components.

Agent component-based development (ACBD) is based on building complex software systems by reusable and simpler agent components. The ACBD is expected to be capable of reducing development costs and improving the reliability of the entire software system. A (software) agent component is a software component [41][51] that takes advantage of both software agent and (reusability, economy, (user) modifiability, extensibility) component techniques, and overcomes the difficulties of the method of the single agent or component development. The interface of an agent component is a component's part that defines explicitly and semantically its access points (e.g., communication protocol), which make the agent services available to the environment and accessible by other agents, and explicit context dependencies.

The analysis of agents has progressed to defining functionalities and related services, but agents must be converted into a component format. The agent component architecture serves to document internal structures and relations for reusing software agent components [74]. These properties are used to construct customized agent models. A repository that enables agent reuse may store agent components that belong to numerous projects within a given application domain. Knowledge derived from the application domains will be used to organize and manage the agents.

2.2. Recommendation System

The quickly growing quantity of information on the web makes it difficult for users to find the desired information. Search results are dependent on users' familiarity with suitable search tools to find information. Search engines aim at facilitating the search process but they have deficiencies such as the lack of personalization. In recent years, as the volume of content and the number of services on the web grow rapidly, web personalization and recommendation

systems (RSs) are in more demand [4]. In conformity to the characteristic and requirement of diverse users, personalization initiative services analyze and classify information resources, and further recommend users information according to their demand and preference, alleviating users from information overload. There are various types of recommendation systems [21]: using recommendations from other users or admitting queries from a user exploiting knowledge about his needs, and based on behavior patterns search profiles and content information in spite of making personalized suggestions. Producing personalized recommendations, these systems overcome the drawback of search engines.

The challenges people face in navigating enormous information spaces is similar to those faced by software engineers looking for one artifact among hundreds of libraries. Recommender or recommendation systems have been proposed as a solution to this well-known research issue in the information retrieval and collaborative systems fields [21]. Recommendation systems commonly recommend similar items, documents or people.

Recommendation systems for software engineering (RSSEs) are emerging to support software engineers in various activities, from reusing code to writing useful bug reports. RSSEs involve user profiles (any combination of user-specified and learned aspects) and proper filtering techniques to provide guidance and assistance to users selecting and navigating by the artifacts estimated to be the more relevant of their interest, overcoming the overwhelming feeling when facing a large artifact volume source, such as the web or organizational repositories for example. Even if a user is willing to reuse an artifact, perhaps he may not be able to locate it in the artifact repository. When artifact repositories increase in size, it becomes a challenge for users to remain conversant with all artifacts. Users must learn to navigate large libraries or information sources. If a user believes a reusable artifact, for a specific task or with certain properties, does not exist then the user is unlikely to query the artifact repository. Repositories improve software development productivity. RSs filter information that potentially may match the user's interest. User profiles play an important role in the success of recommendation process since they model and represent the user's preferences.

We posit that the semantic information from the user context has considerable impact on the performance of knowledge-based recommendation

systems. In traditional RSs the user profile is based on keywords, which involves to some extent only those items that match specific keywords in the user's preferences [77]. Nevertheless, several problems with keyword-based profile representation methods arises such as losing a lot of valuable information, inaccuracy in the recommendation artifacts, and user dissatisfaction. One approach to overcome these problems is to extract and use semantic-based information from the domain and incorporating them within the stages of the personalization process. In some approaches the knowledge comes from users preferences, but in our approach it comes from ontological knowledge about the stored software artifacts, provided by the repository users themselves.

2.3. Variability Model

Variability is the ability of a system or artifact to be modified, extended, or configured to be used in a certain situation [62]. Variability modeling becomes necessary when users derive new specifications for further artifact implementation from the collection of commonalities and variations related to a specific system family. The purpose of analyzing of commonalities and variations is strategic reuse to identify common parts and develop them for reusability. The variability in such domains distinguishes the members of a family from each other and needs to be explicitly modeled and separated from the common parts. Variability management deals with these differences by handling the introduction, use and evolution of the variability.

Common and variable features are often represented by means of the widely adopted feature model [62]. A feature model is a hierarchical model where each characteristic of the domain relevant to a stakeholder becomes a feature. Thus, features and their dependencies are defined by a feature model. Relationships among a feature and its sub-features are categorized as mandatory (sub-feature is required), optional (sub-feature is optional), or (at least one of the sub-features must be selected), and alternative (one of the sub-features must be selected) [62].

Based on these characteristics, variability model offers good conditions for its applicability to the particular context of the agent-based artifact repository. Since various software agents, in different domains, have common and variable

features, the variability model allows the derivation of new agent components that are adapted according to the individual needs of each software engineer.

2.4. Information Retrieval Model

There are enormous volumes of information to which accurate and fast access turns ever more difficult. Balancing efficiency with information retention is a serious challenge in information retrieval. To overcome this problem, information retrieval systems are designed to locate and present information relevant to an expressed information need.

Information retrieval [4] deals with the representation, storage, organization and retrieval of items that contain the information desired (e.g., text documents, multi-media objects). Information retrieval (IR) systems allow users to look for information in a collection of sources through queries, and using the query it retrieves information that must be relevant to the user. The main goal of an IR system is executing searches and returning as many relevant items as possible according to user' needs and minimizes false positives.

For the IR to be efficient, the items are typically adapted into a suitable representation. An IR model specifies representations used for items and queries, and how they are compared. Generally an IR model is a conceptual framework that describes how the items should be modeled, how users describe their needs, and a ranking function. Ranking is fundamental to search for determining how results are retrieved and ordered. For example, the vector space model [50][4] describes documents and queries as vectors and the ranking function is based on the distance between the vectors that describe documents and queries. There are other IR systems like the keyword-based that are limited in its ability to distinguish between relevant or irrelevant texts in which these terms are just mentioned occasionally. This can be the case when there are terms describing the same object. In keyword-based approach, there is a lack of domain-knowledge associated with its queries. As keyword-based search does not guarantee relevance in meanings, semantic search has attracted huge and growing interest to improve the accuracy of ranking.

Information retrieval, in the context of software engineering, consists of recovering software artifacts that respond to the user' requirements. Its difficulty comes from the heterogeneity of the artifacts and the diversity of the application fields.

A component retrieval method can be described from the following phases [76]: component representation, component query (users' requirements) specification, and component retrieval process. The component retrieval method, named free-text-based retrieval method, comes from the information retrieval community. In this technique, the components are represented as free-text-based documents, while the component query is described using keywords [51]. The process of retrieval looks up the keywords in all component description documents. The components with most matched keywords will be returned. Its limitation consists in the ability to distinguish between relevant and irrelevant texts due to the synonymy and polysemy. The synonymy is when there are numerous terms to describe the same concept. The system will only retrieve those items that refer to the concept by the same term used in the query. The polysemy is when a word has multiple meanings, i.e., expressing different concepts. In this case a query might lead to the retrieval of items which deal with concepts foreign to the subject of interest. To go beyond this obstacle, an IR system needs another structure for the data with which it works, e.g., a structure based on concepts instead of mere keywords. There have been proposed the use of thesaurus to extend keywords, by including their synonyms and antonyms, to get more relevant components. Moreover, domain knowledge is also used to extend initial keywords to get more semantically relevant components.

In place of free-text-based component and query descriptions, the following four types of retrieval methods represent components and specify queries using structural information from dissimilar perspectives.

The pre-enumerated vocabulary method uses a set of pre-defined vocabularies to express components and queries. Thus, recall and precision are increased at the cost of the flexibility to describe components and specify component queries.

The signature matching method [85] describes components and queries using signatures which specify the structure description and interfaces of components. The signatures matching approach is based mainly on both matching

and type transformation techniques. For instance in the context of the development by objects, a component is often composed of various classes which contain a set of method signatures, types and variables. Consequently, a component can be represented by a set of signatures. This technique is based on the hypothesis that the set of signatures accepted by the system is known. The signatures include functions and simple or complex types. This technique is interesting if the application engineer knows in advance the form or the name of a method belonging to the retrieved component. It will improve the productivity for a user who intervenes in a late phase of the information system development cycle and handles hundreds of software components. The weakness of this approach is its lack of domain and searching context information.

The behavior-based retrieval method [2][61] is interested in the dynamic feature of the software components, by analyzing their behavior. The component execution trace, obtained by using random values of parameters, is a relevant criterion to classify and retrieve the components. Components take the form of executable codes, and queries are represented by a set of input samples and their desired outputs. The retrieval proceeds by selecting samples, and executing components using the selected samples. The components that satisfy the desired output are retrieved. Having a specific behavior to retrieve a component, it is necessary to establish an order relation on the components' behaviors. The query is presented as a program which systematically calls a subset of the operations of each component in order to compare them to the required behavior. This method is designed for executable software components and has low efficiency because of long execution time. Its advantage is to allow finding simple components (classes) and complex ones (graph of classes).

Other method in this category is faceted selection [86]. This approach predefines a set of dimensions, called facets, which are used to classify components from diverse perspectives. Each facet represents information that allows identifying and selecting a component. Users can find desired components by searching down the facets. This method is getting increasing attention due to it takes domain knowledge into account when designing facets. But there exists a design embarrassment: if facets are designed very simple or few, there will be too many components in final categories, which will ask users to select further manually. On the other hand, if facets are designed too complex, it is hard for

users to understand them and hard for designers to classify all components into different categories. Besides, the faceted selection method in essence uses the exact matching process. Nevertheless, it is very difficult to get the appropriate components through exact matching because of the universal differences between component requirements and components descriptions. This technique can be easily enlarged and most flexible giving good results when providing to select good facets and give good terms for the components' indexing. Nevertheless, it requires a manual indexing that can be expensive.

All these retrieval methods have a common statement: users can define their component queries, and the retrieval system can find one or a few appropriate components according to users' queries. However, this assumption is not always realistic. Users frequently lack clear ideas about what they need while they begin searching for components and usually cannot define the queries accurately. Users need a retrieval system to guide them refining their queries. For that reason, an efficient component retrieval system should be able to support partial matching, select components based on syntactical and semantic similarities, and guide users to refine their component query incrementally.

2.4.1. Scoring, Term Weighting and the Vector Space Model

Rather than a set of items satisfying a query expression, an information retrieval system returns an ordering over the (top) items in the set with respect to a query. A particular ranking of search results is influenced by some factors, i.e., basic units (tokens) that represent the items (preprocessing like stop word removal or stemming); assigned weight marking the importance of these tokens for the items; the relevance score calculated that determines the ranking (how well an item and the query match).

In contrast to the Boolean model where the presence of terms in documents is binary [50][4], a weight can be assigned to a term to express its importance for a specific document. A commonly used term weighting method is *term frequency – inverse document frequency* [50], *tf-idf*, which assigns a high weight to a term if it occurs frequently in the document but rarely in the whole document collection.

On the other hand, a term that occurs in all documents has hardly any discriminative influence and a low weight is given, which is typically true for stop words. For calculating the *tf-idf* weight of a term in a particular document, it is necessary to know (i) how often the term (or its stemmed or case variant forms) occurs in the document, called term frequency, *tf*; and (ii) in how many documents of the collection it appears, called document frequency, *df*. Taking the inverse of the document frequency, *idf*, the weight of the term in the current document is calculated by multiplying *tf* by *idf*. But in practice [50], the *idf* is calculated as follows:

$$idf = \log_{10}\left(\frac{N}{df}\right)$$

where the quotient is the total number of documents, *N*, by the document frequency in order to scale the values.

The *tf-idf* values are used to create vector representations of documents. Each component of a vector corresponds to the *tf-idf* value of a certain term in the corpus. This kind of representation in a common vector space is the vector space model [50][4]. Given two vectors that represent two documents, to know if both documents are similar, the similarity of vectors is calculated making use of the cosine similarity measure, as follows:

$$cosine\_similarity(d_1, d_2) = \frac{\vec{v}(d_1) * \vec{v}(d_2)}{|\vec{v}(d_1)| * |\vec{v}(d_2)|}$$

The cosine similarity is in the range [0…1] since the term frequencies (*tf-idf* weights) cannot be negative. Therefore, if two documents do not share at least a word the similarity value is zero, while documents that share a similar vocabulary get higher values, up to one in the case of identical documents. If a query is considered as a short document, it is possible to create a vector for the query, which can then be used to calculate the cosine similarities between the query vector and those of the matching documents. To conclude, the similarity values between query and the retrieved documents are used to rank the results.

2.4.2. Navigation Model

Navigation is a major information access mechanism for most users [44], which determine the usefulness or relevance of the information currently displayed and browse its associated links called relations. It exploits the implicit or explicit relations existing among artifacts. The user navigates inside a graph using the relations. The semantics associated with these relations guides the user according to his needs to choice the best path in the graph. Navigation is a convenient pattern of artifact repositories due to it exhibits a list of software artifacts that could probably be used in the current development situation, so users do not need formulate clearly-defined requirements for reusable artifacts. This automation of the artifact location process not only reduces the search of artifacts through rich information space so, users can easily browse and choose the desired artifacts, but also enables users to get familiar with the available artifacts in the repository and use them whose existence they do not even anticipate.

2.5. Semantic Web

The semantic web envisions a WWW in which data is described with rich semantics and applications can pose complex queries. The semantic web has recently increased the interest on using conceptual models to specify explicitly the semantics regarding knowledge about software artifacts. Taxonomies and ontologies are types of conceptual models which can be applied to describe a domain of discourse, modeling it as a set of concepts and relations. Though taxonomies focus on hierarchical relations between concepts, ontologies may exhibit more complex relation types, such as part-of and associations.

2.5.1. Ontology

Currently in the semantic web application development, ontologies have emerged as a powerful tool to enable knowledge representation and knowledge sharing, by a community of a target domain in an explicit and formal way, and to achieve semantic interoperability among heterogeneous distributed systems.

Ontologies have gained wide acceptance as a model of information in a given domain that can be used for many purposes, including database design and integration [20], information retrieval and extraction [20], information interchange on the WWW [10], and in enterprise integration solutions [20] (they provide a shared and common understanding of data that exist within an application integration problem domain). Also, ontologies would enable knowledge reuse and a standardized model for the cataloguing of software artifacts. According to the IEEE Standard Upper Ontology Working Group[1], an ontology is "a set of concepts, axioms, and relationships that describe a domain of interest." Therefore, they are powerful knowledge independence representation structures, in the case of this work especially useful for the specification of reusable agent-oriented software artifacts and their relations since they establish a joint terminology among members of a community.

Because their characteristics such as clarity, formality, applicability and adaptability, ontologies provide a terminology and a controlled vocabulary that can be shared by everyone involved in a development process and useful for the retrieving of reusable artifacts. Consequently, the limitation of natural language's imprecision can be adequately managed and then the complexity of formal methods is reduced.

The use of ontologies offers some other benefits, for example over syntactic solutions in numerous aspects including their semantic definition of information enables a classification of knowledge and implementation of rules, explicit description of processes and high level software abstractions [47], relationships and data to discover and navigate of resources, disambiguate functionalities and products avoiding conceptual and terminological confusion, capacity of inference (a machine-learning process borrowed from artificial intelligence) and flexible querying capabilities in this knowledge. The capacity of inference of a system consists on finding hidden information deriving new data and knowledge from existence data, that is to generate new statements based on the existing knowledge. The inference classifies concepts, checks consistency of the knowledge base, can find much connotative knowledge and inconsistencies about

[1] http://suo.ieee.org/

the knowledge base. The inference can be presented in repositories, e.g., RDF [67] and OWL [58] repositories, which are free and without needing to programming.

Ontologies also bring advantages in the information retrieval area. One of the major challenges in the development of search engines is the ability to differentiate between relevant and irrelevant results. This differentiation is disadvantaged generally by factors of polysemy (terms have diverse meanings) and synonymy (several words describe a specific concept). These factors can be reduced with the use of ontologies seen as vocabularies hierarchical representation to describe knowledge domains. In addition, inference mechanism interprets queries to enrich and make more efficient the information retrieval with semantically annotated information.

There are other advances of ontologies, for instance respect to databases. Ontologies store data that features the subject of data processing and information system environment within the model that provides rich semantics, reflect semantic characteristics of data that are partially reflected in the database semantics, analyze requests and transforms them in the queries. Meanwhile, a database stores data that is the primary object of data processing within the model that is the most beneficial for performance and ease of maintenance on large data volumes, reflects conceptual peculiarities of a particular domain, and is task-oriented and execute queries. Ontologies are standardized, meaningful, machine-readable, platform independent, programming language independent, can be shared in the web and reusable though and read by anyone freely, the information is modeled in a more flexible way than using tables, this information can be later classified based on the properties of the instances, and the knowledge base can be queried including semantic information.

Ontology-based tools can provide advanced services for applications such as applications “smart” databases, search and semantic indexing (in the web), tools for decision support, applications for e-commerce, software agents, etc.

Ontology and agent technology are now fundamental assets to the semantic web. Ontology provides shared common understanding of a domain and facilitates content-based access, interoperability among systems and communication across the web. Software agents can parse, understand and reason about information available on the semantic web resources in an attempt to aim its objectives. The combined use of ontology and agent technology enables the sharing and

exchanging of heterogeneous, autonomous knowledge sources in a robust, adaptable and extensive manner [10].

2.5.2. Taxonomy

From the user's point of view, the lack of familiarity with the vocabulary is also a drawback in using a retrieval system effectively. In this context, the use of taxonomies is considered a promising solution for query specifications. Taxonomy [10] is "the classification of information entities in the form of a hierarchy, according to the presumed relationships of the real-world entities that they represent."

The use of taxonomies in the software engineering helps search engine as an alternative to improve the search result quality. Taxonomies define attributes to the terms as well as identify, name and classify them being showed in form hierarchy. Artifact modifications create new versions that must be catalogued for future reuse. The effort required for future retrieval increases, since users must identify the artifact with the best fit among a number of similar artifacts.

2.6. Subscription Service

RSS (Really Simple Syndication) is a way of delivering and consuming up-to-date information on the web, which should be exploited to effectively promote their produced content. RSS helps users to save time by not visiting each site individually, to select appropriate feeds and keeping them up-to-date. RSS is written in XML language for syndicating information items on web.

Libraries, in general, are ideal candidates for adopting and using RSS to their advantages as information producers (libraries maintain their web sites providing information about its services, new additions, announcements, and some other type of information), information gateways and information consumers.