Self Describing Components: Searching for Digital Artifacts on the Web *

André Santanchè[1] **, Claudia Bauzer Medeiros**[1]

[1]Institute of Computing – University of Campinas – UNICAMP
CP 6176, 13084-971 Campinas, SP, Brazil

{santanch,cmbm}@ic.unicamp.br

Abstract. *The Semantic Web has opened new horizons in exploring Web functionality. One of the many challenges is to proactively support the reuse of digital artifacts stored in repositories all over the world. Our goal is to contribute towards this issue, proposing a mechanism for describing and discovering artifacts called Digital Content Components (DCCs). DCCs are self-contained stored entities that may comprise any digital content, such as pieces of software, multimedia or text. Their specification takes advantage of Semantic Web standards and ontologies, both of which are used in the discovery process. DCC construction and composition procedures naturally lend themselves to pattern-matching and subsumption-based search. Thus, many existing methods for Web searching can be extended to look for reusable artifacts. We validate the proposal discussing its implementation for agro-environmental planning.*

1 Introduction

The search for efficiency in software development has prompted intensive research in reuse and documentation practices. The same goals and practices have propagated to the area of content design and management. The Web has accelerated such initiatives: IT professionals need new kinds of tools and techniques to retrieve the appropriate digital artifacts from repositories all over the world. This presents challenges both in specification and description, as well as in good searching mechanisms.

As a result of these efforts, there is an increase in the interchange of reusable artifacts (content and software), assembled inside standard "containers" – the *packages* – and stored in package libraries [CCSDS 2002]. We define a package as a structure that delimitates, organizes and describes one or more pieces of digital content suitable for reuse. The term *digital content* is used from now on to denote any content represented digitally – e.g., pieces of software but also texts, audio, video, and so forth.

However, while the size of package libraries grows, effective reuse depends on the ability to discover artifacts for given requirements. Klischewski [Klischewski 2003] observed that there is a variety of resources, like fine-grained information elements, multimedia items, services, or user related objects, which are meaningful for users. Therefore, they are candidates for semantic markup using Semantic Web standards, providing a more

*This work was partially financed by UNIFACS, by grants from CNPq and FAPESP, and projects MCT-PRONEX SAI, CNPq WebMaps and AgroFlow, and Microsoft eScience grant.

semantic way to search and use them. This observation was made in a e-Government context, but can be extended to the general reuse context.

Semantic Web efforts have addressed two directions: a common syntax and semantic to exchange data; and a common syntactic and semantic infrastructure to provide interoperability between processes. In the former direction, the initial approach was document oriented, through annotations using RDF and OWL and pointers based on URIs connected to Web documents. As pointed out in [Klischewski 2003], there is a wider universe of "annotatable" artifacts on the Web, formed by "annotatable" sub-parts whose organization depends on the artifact's nature. The challenge is how to associate Semantic Web annotations to artifacts and their subparts, in spite of their heterogeneity.

A similar challenge is faced by the second direction to describe different kinds of process entities, meant to inter-operate. In this case, the heterogeneity of process entities is hidden behind a standard interface. Here, Semantic Web-based standards (WSDL and OWL-S) are used to describe process functionality and details of its working activities as a composition of sub-processes.

Another challenge in this scenario is to build new products that properly combine and reuse pieces, in spite of their diversity. Developers in this new scenario will not be just computer science experts, thus requiring new models and tools [Mørch et al. 2004]. Moreover, reuse requires finding the adequate pieces of digital content, and therefore for new means of describing, storing and retrieving these pieces.

WSDL and OWL-S are meant to describe process entities, and thus indirectly associate a *provided functionality* with process entities (e.g., a video player software *plays* video). However, one can also envisage the description of the *potential functionality* associated with the nature of any other digital content (e.g., a video content can be *played*). These functional annotations support finding the appropriate artifacts on the Web.

This paper contributes towards this direction. We propose a unified model to build reusable digital artifacts. It can be used both for content design (content-centric approach) or for software development (process-centric approach). In our model, each piece to be reused is encapsulated inside a unit named *Digital Content Component* (DCC). Furthermore, our model expresses both the provided and potential functionalities via an interface associated to any digital artifact. This functionality-based description provides a richer semantic way to annotate any kind of digital content, hiding its heterogeneity behind a standard interface. Semantic Web standards are adopted in many aspects of DCC specification and Web service standards are adopted for the interface specification.

More specifically, the paper focuses on the technique used to specify DCC interfaces using OWL and OWL-S respectively. As will be seen, these semantically enriched specifications enhance the possibility of reusing content. Moreover, they help discovering DCCs that are suitable for a given product construction in two ways: the functionality description is used to refine the search procedure, and the descriptions associated with DCC operations are used to discover useful DCC subproducts "hidden" within a DCC. For instance, images may provide pixels (e.g., a pixel inside a map) or videos may provide a set of frames (e.g., a commercial from a film), and so on. The issues discussed here are presented by means of a practical example.

The remainder of the text is organized as follows. Section 2 details DCCs and

their specification based on OWL and OWL-S. Section 3 presents our three step procedure for DCCs discovery, based on their OWL metadata and interface specifications. Section 4 presents how DCCs can be connected and assembled to form an application and the role played by the interface specification. Section 5 considers related work, showing how DCCs combine and generalize distinct reuse approaches. Section 6 presents the conclusions.

2 Specifying DCCs

The specification of DCC considers two issues: clear separation of content and interface specification, to support reuse; and use of ontologies for semantic annotation, to help find appropriate DCCs and to match their connections. This section presents DCCs using as background a real example for agricultural planning. Assume that experts want to forecast the evolution of a given coffee plantation under certain weather conditions. Given these as input, together with coffee plant geographic location, the output shows how the plants will evolve. This result is seen by means of an animation that simulates the growth of a plant for the input conditions provided.

2.1 An overview of DCC

A DCC is specified and stored as a unit composed by four distinct sections: (i) the content itself, in its original format; (ii) the declaration, in XML, of an organization structure that defines how components within a DCC relate to each other; (iii) a specification of the DCC interfaces, using adapted versions of WSDL [Chinnici et al. 2004] and OWL-S [Martin et al. 2004]; (iv) metadata to describe functionality, applicability, use restrictions, etc., using OWL [Smith et al. 2004].

We differentiate between two kinds of DCC – process and passive DCCs. A process DCC is process-centric: it encapsulates any kind of process description that can be executed by a computer (e.g., sequences of instructions or plans). A passive DCC is content-centric (e.g., a text or video file) and its interfaces define how its content can be accessed.

DCCs are assumed to be stored in repositories on the Web. Interface and metadata sections are used to help retrieve the appropriate DCCs from the repositories. There is furthermore a DCC infrastructure that comprises an architecture to assemble DCCs into a desired product. For more details on DCCs, see [Santanchè and Medeiros 2004].

Ontologies play a fundamental part in DCC description and semantic management. According to [Cullot et al. 2003], there are two main kinds of ontologies: descriptive and taxonomic. A descriptive ontology resembles database schemas. Its concepts are interconnected by many kinds of semantic associations, and its purpose is to represent the intended domain as much as possible. A taxonomic ontology is used as a basis for vocabulary alignment. Its structure organizes terms into generalization/specialization hierarchies, and semantic links to express synonymy, composition, and so on.

Taxonomic ontologies are useful in information sharing activities [Cullot et al. 2003]. We adopt them in DCCs to disambiguate the meaning of DCC metadata and interface specification. More specifically, we postulate the need for specific ontologies that define valid kinds of DCC and of terms used in defining DCC interfaces. Fig. 1 shows diagrams that represent parts of two taxonomic ontologies, used

by our examples, and whose hierarchical relations will be explored in DCC semantic relationships and search procedures. White-filled circles represent classes. Lines with a diamond in one extremity represent subclass relationships, e.g. `Rain` is subclass of `Precipitation`. Dashed lines indicate that some intervening nodes were ommitted for simplicity.

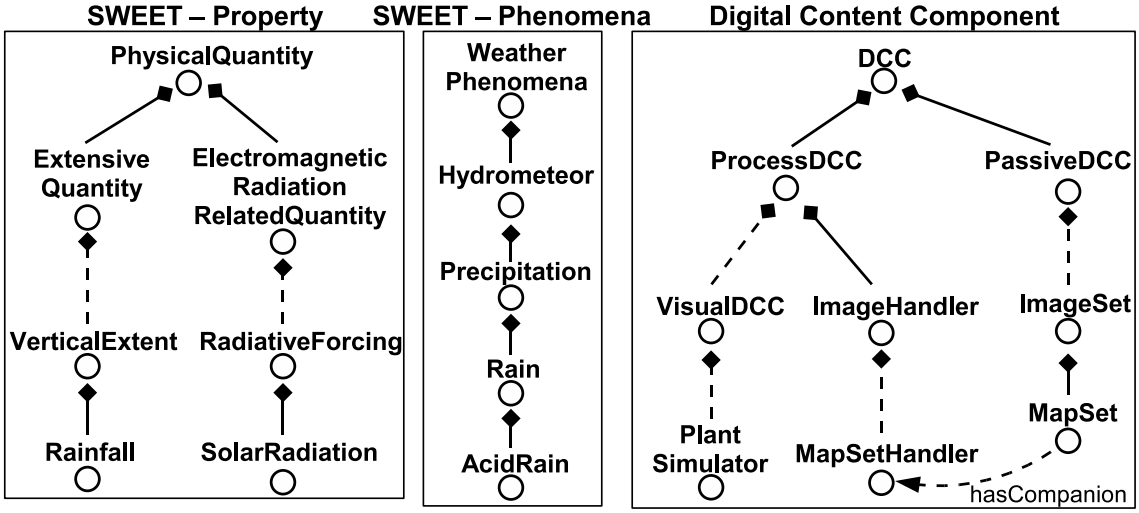

Figure 1. Ontologies used by rainfall map component.

Our example concerns managing, creating and reusing content for agriculture applications. DCC discovery and reuse require domain semantics – in this case, the taxonomic ontology called SWEET – Semantic Web for Earth and Environmental Terminology [Raskin and Pan 2003]. Fig. 1 shows two fragments of SWEET. The fragment at the center concerns a taxonomic hierarchy about Rain, while the left fragment describes physical measurements. The right fragment is part of our ontology constructed to classify DCCs according their functionality. Each class in this ontology corresponds to a DCC type, and each DCC is an instance of a class.

2.2 The Rainfall Map Content Component – Content Centric Approach

Our application requires combining rainfall and solar radiation data with coffee plant growth simulation. We show how this can be done by first creating the DCCs and then composing them.

Fig. 2 shows an example of a partial representation of a passive DCC. This component encapsulates a temporal series of images containing one year of rainfall data for São Paulo state. Each image is visualized in a map and represents the average rainfall distribution in one month (i.e., each pixel contains the average rainfall value for the corresponding region). The internal organization structure of the component, a set of twelve images (the content), is described in XML.

Both metadata (in OWL on top) and interface (in OWL-S displayed around the organization structure) are presented using a simplified version of RDF-like Directed Labelled Graph (DLG). Metadata and interface parameters are associated with ontological terms.

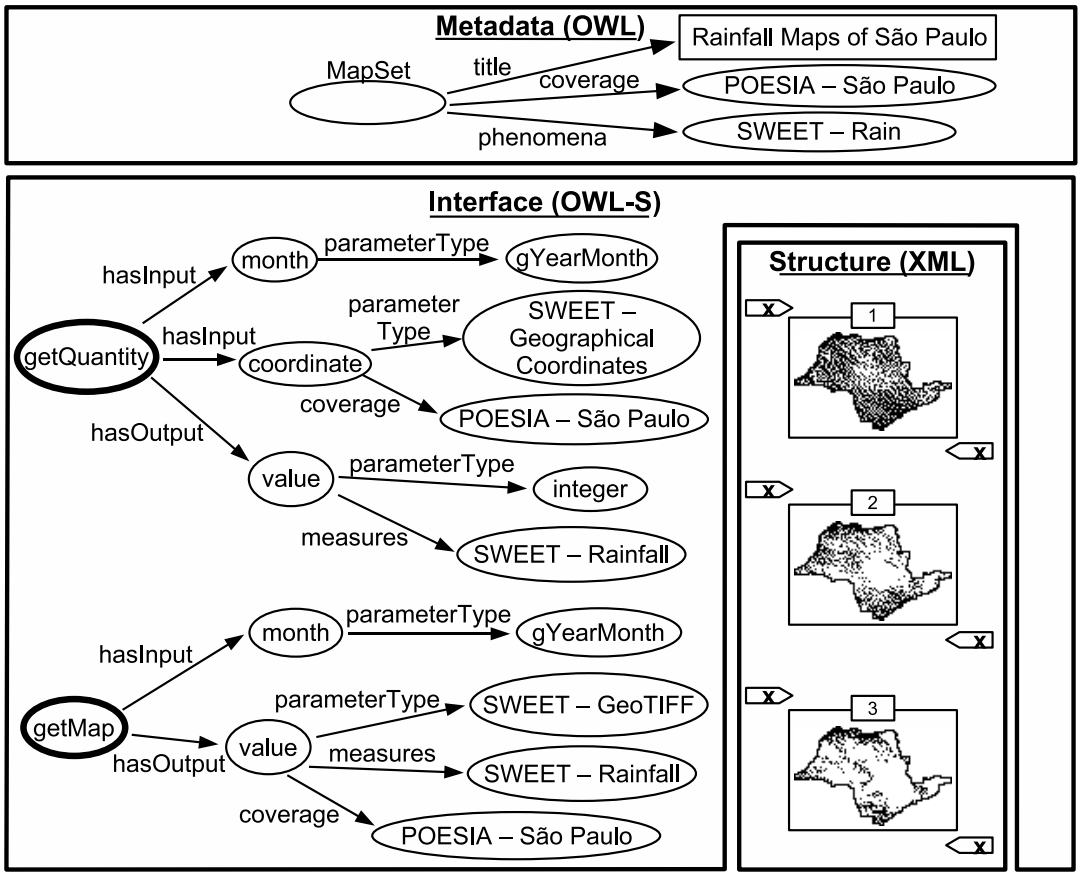

Figure 2. Rainfall map content component representation.

In the metadata section there is a reference to the DCC ontology presented in Fig. 1. It shows that the DCC is an instance of the $MapSet$ class, with three property values: $title$, $coverage$ and $phenomena$. The values of $phenomena$ and $coverage$ are respectively related to SWEET and to the POESIA spatial ontology [Fileto et al. 2003]. The latter is a spatial ontology specific to Brazilian spatial unit organization.

The interface section presents operations using OWL-S $ServiceModel$ class hierarchy [Martin et al. 2004]. It defines two operations (atomic processes in OWL-S): $getQuantity$ and $getMap$. The $getQuantity$ operation returns a value of a pixel inside a map image, given parameters $month$ and pixel $coordinate$. The $getMap$ operation returns a map image for a given $month$ parameter. These atomic processes, which receive one input message (comprising all input values) and return one output message, correspond to WSDL request-response operations [Martin et al. 2004]. To simplify the explanation we will use the same names of OWL-S atomic processes to refer to WSDL related operations.

These operations illustrate how the descriptions can be connected with taxonomic ontologies. Following the OWL-S model to describe processes, each process parameter has a $parameterType$ which specifies the class or datatype for that parameter [Martin et al. 2004]. Notice that many ontologies may be needed to properly specify a parameter. For instance, the $coordinate$ input parameter has a type description associated with SWEET, but its domain is defined by the $coverage$ property, in POESIA, here denot-

ing that the only valid coordinates accepted are those from within the state of São Paulo. The output parameter of the $getQuantity$ operation is an integer value. The additional $measures$ property of the SWEET ontology defines the nature of measured value. Notice that we extended the OWL-S schema to enhance parameter description with additional semantics. OWL-S specifies the need for type characterization ($parameterType$) which we improved by adding semantic parameter descriptors (e.g., coverage, measures). The parameters of the $getMap$ operation work the same way.

This extension to OWL-S to organize components is similar to the *faceted* method, borrowed from library science by Prieto-Díaz [Prieto-Díaz 1989] to classify software components. In contrast to the traditional *enumerative* method adopted by library science, which uses a classification tree to organize components in categories and sub-categories, the *faceted* method describes components by a set of attributes (named facets); each facet is specified by setting a pertinent term value. We used the RDF/OWL description approach to attach a set of descriptive property values (facets) to each parameter.

It is important to note that the DCC of Fig. 2 is *passive* – does not embed the program code to execute these operations. The focus in this kind of component is in the content (i.e., the maps themselves), and the operations define how this content can be accessed. Since the program code for the operations cannot be embedded in a passive component, interface operations are implemented in a *companion component*. The association between the passive component and the companion is achieved with help of semantic information given by terms of the DCC ontology. The companion for the $MapSet$ component is the $MapSetHandler$ (see Fig. 1). $MapSet$ has a property value pointing to the $MapSetHandler$.

2.3 The Coffee Plant Simulator Process Component – Process Centric Approach

Fig. 3 shows an example of a partial representation of a process component. This is a software component that graphically simulates the growth of a coffee plant for a given coffee strain, and specific weather and location conditions. Its internal structure organizes Java binary code classes, which implement the simulator software, and related files.

To execute its job, the simulator DCC requests services from external DCCs. There are three processes, declared in the interface, for the requested services: $rainfall$, $solarRadiation$ and $growthRate$. They actuate in two stages, being thus composite processes. First they request a service by sending a message, containing their output parameters; next they receive the result of the solicited service in a message, whose content must match their input parameter. This kind of composite process corresponds to a WSDL solicit-response operation [Martin et al. 2004].

The simulator DCC uses $rainfall$ and $solarRadiation$ processes to request weather data, essential to estimate the coffee plant $growthRate$. In more detail, $rainfall$ provides parameters $month$ and $coordinate$ (whose semantics and types are defined ontologically) and receives back from an appropriate service a $value$ whose meaning and type are likewise defined. The same applies to the $solarRadiation$ process. Additionally, the simulator DCC declares the $start$ process, which is atomic and corresponds to a WSDL one-way operation.

We point out two further characteristics of DCC construction. First, the interface specifies processes. These may be operations implemented locally, or *other compo-*

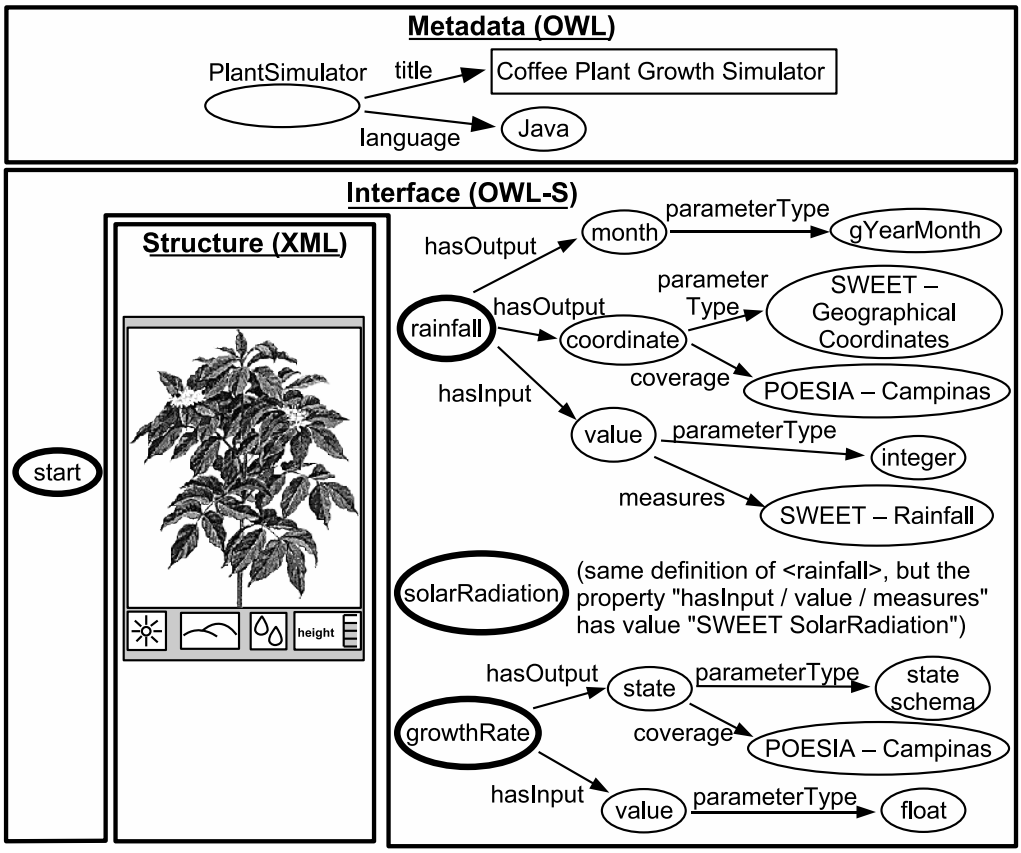

Figure 3. Coffee plant simulator process component representation.

nents that have been built (reused) into the simulator. Second, parameter semantics define process (and component) semantics. Notice that in the $rainfall$ and $solarRadiation$ operations the $coverage$ of the $coordinate$ output parameter is $Campinas$: this component was built to simulate the coffee plant growth under Campinas city weather conditions, therefore it will only process values in this coverage. These constraints are used in a discovery process.

3 Discovering DCCs

A key aspect of our proposal is how a designer discovers DCCs for reuse. DCCs' metadata and interfaces are specified in OWL/OWL-S, which can be used in component indexation and searching. Domain ontologies can help in this task in three ways: (i) they organize DCCs in taxonomic trees that can be navigated in the discovering process; (ii) ontology concepts are used to help query construction, to disambiguate terms and find synonyms; (iii) ontological relationships are used to rank DCCs based on their similarity with the searched DCC.

In DCC discovery process the designer can navigate through taxonomic trees to search for a DCC. The designer may define values of properties to characterize a desired DCC. Alternatively, the characteristics of DCCs already in a composition guide the search for the new one.

Let us consider a designer that wants to build a composition to graphically simulate a coffee plant growth. He/she starts by the $simulator$ DCC obtained navigating in

the DCC taxonomic ontology and selecting a DCC instance of $PlantSimulator$ class – see Fig. 3.

The next step is to connect the plant simulator instance to a DCC that provides the rainfall average for a given month and coordinate. Looking at the simulator specification, the designer will next query the Web looking for a DCC that supports the `Campinas` $coverage$ of the POESIA ontology, and the `Rain` $phenomena$ of SWEET ontology. As often occurs in this kind of searching process, maybe no DCC exactly matches with the query. The search engine can take advantage of the ontological semantic relationships to find other "most similar" DCCs and rank them depending on their similarity.

Our search procedure follows three steps: (i) metadata similarity-based searching and ranking; (ii) interface searching and ranking via inheritance relationships; (iii) interface matching-based refinement and ranking. Each of these steps will be detailed in the following three subsections.

3.1 Metadata similarity-based searching and ranking

Consider that the designer wants to retrieve a DCC via a query specified using the RDF-like DLG. The first step looks for metadata similarity selecting DCCs whose metadata graph is "similar to" the metadata query graph. For each query property, the search engine verifies if the same property exists in the DCC; if not, it verifies if there is a property in the DCC that is defined as an OWL subproperty of the query property.

The $rank\_similarity$ routine – called by the search engine – defines a value between 0 (no similarity) and 1 (equivalent concepts). $rank\_similarity$ uses ontologies to compare a DCC property to a query property, acting in three directions to determine: equivalent concepts, more general concepts and more specific concepts. The priority order in the ranking is: equivalent, general and specific, and can be inverted depending on the desired results. The search engine sums the ranked values of all properties.

In this comparison, two values A and B are considered equivalent if they refer to the same concept in the ontology (equal URIs), or if they point to two concepts related by OWL equality relationships (`equivalentClass` or `sameAs`). Moreover, A is said to be more general than B if A subsumes B and conversely B is more specific than A. For instance, if B is OWL `subClass` of A, or B is related with A through the $partOf$ property ($B\ partOf\ A$), then A subsumes B. Consider A and B vertices of a graph, whose edges are properties. The subsumption relationship between A and B is a path formed by one or more edges. Therefore, the similarity rank value between A and B is inversely proportional to the number of edges which connect A and B in a subsumption relationship.

Let us return to the designer whose query is for DCCs with `Campinas` $coverage$ of POESIA and `Rain` $phenomena$ of SWEET. Assume that two DCCs obtained in the query response declare the following metadata: **(DCC1)** the São Paulo rainfall map, presented in Section 2.2; **(DCC2)** Campinas satellite images for days of acid rainfall, which declares a `Campinas` $coverage$ and `AcidRain` $phenomena$.

DCC1 satisfies the search because its metadata has a $phenomena$ concept equivalent to the query parameter on this concept, and because its $coverage$ metadata relates to the `São Paulo` concept, that in POESIA ontologically subsumes the query predicate

on Campinas: DCC1 covers a more general spatial surface than the one specified in the query. Thus, it includes the queried Campinas coverage. DCC2 metadata has an equivalence relationship on the query for the $coverage$ concept (Campinas) and a subsumption relationship on the AcidRain concept (since in SWEET Rain is a superclass of DCC2's AcidRain – see Fig. 1). This means that this DCC produces a kind of rainfall average stricter than the one specified in the query. This result can be useful if the designer uses a generic concept to express a set of desired sub-concepts, for instance, if the designer wants to search for maps of any Brazilian state, he/she queries for Brazil $coverage$ and expects to receive stricter coverages. For this reason the order of generalization/specialization in similarity ranking can be inverted and depends on the search context.

3.2 Interface searching and ranking via inheritance relationship

The query in the previous section can be refined by specifying, besides metadata, the kind of output expected from the desired DCC. In this case, this output has to match the input of the rainfall operation of the $simulator$ DCC: a value with integer $parameterType$, whose semantics are defined by SWEET Rainfall. The similarity procedure to find and rank similar interface specifications is the same of the previous section.

Assume that this refined query returned another DCC – DCC3 – that will be added to the previous result (DCC1 and DCC2). **DCC3** is a software component that provides access to a remote weather repository for Campinas, and which declares an output with an integer $parameterType$ and SWEET PhysicalQuantity for the $measures$ property.

DCC3 satisfies the search because its output description has an equivalence relationship on the query for the $parameterType$ concept (integer) and a subsumption relationship on the PhysicalQuantity concept (since in SWEET Rainfall is a subclass of DCC3's PhysicalQuantity – see Fig. 1). Notice that this step enhanced the searching process, finding additional DCCs based in finer-grained criteria.

3.3 Interface matching-based refinement and ranking

The interface specification is again used to further refine the search. For example, it is necessary to verify if the components selected in the previous queries have operations which match with the $simulator$ component. The designer refines the ranking, asking which of the three DCCs have an interface with an operation that matches with the $rainfall$ operation of the $simulator$ component. Here, interface matching is used to discard those DCCs whose interface does not match the request. While the previous step uses interface descriptions to increase the DCC candidates, this step can reduce the set of candidates.

Let X and Y denote inputs declared in the interface of two DCCs and consider the meaning "equivalent", "more generic" and "more specific" of Section 3.1. We define the follows relationships: (i) X EQ Y if for each property of X, Y has the same property with an equivalent value; (ii) X GE Y if for each property common to X and Y, the value of the X-property cannot be more specific than that of the Y-property, and conversely Y SP X. Let now Q be the DCC specified in a query, and inQ and outQ the set of all its inputs and outputs respectively. Let S be a DCC and inS and outS the set of its all inputs and outputs.

We define four levels of interface matching: **exact:** If `inQ` EQ `inS` and `outQ` EQ `outS`; **plug-in:** If `inQ` SP `inS` and `outQ` GE `outS`; **wider:** If `inQ` GE `inS` and `outQ` SP `outS`. **fail:** Not classified in the previous degrees.

The **plug-in** match is a simplification of the one proposed by Zaremski and Wing [Zaremski and Wing 1997]. This kind of match guarantees that the `inS` input domain is a superset of the `inQ` input domain, hence, the `S` DCC can deal with any input of the domain specified in `inQ`. The `outS` output domain is subset of `outQ` output domain, hence, any output generated by the `S` DCC is within the expected results. In a nutshell, the `S` DCC can be plugged in any system where `Q` is required, and will not compromise the system functioning with an unexpected behavior. On the other hand, `S` is not equivalent to `Q`.

The **wider** match is the inverse of the plug-in match. This is the interpretation made by Paolucci et al [Paolucci et al. 2002] to the Zaremski and Wing plug-in match. Here, since the `outS` output domain is a superset of `outQ` output domain, it is expected that the `S` DCC can fulfill any output requested by `Q`. Analogously `inQ`, which is a superset of `inS`, can fulfill all input needs. However, it cannot be guaranteed that the `S` DCC will work properly if, for example, `inS` receives an unexpected input.

Returning to our example, the interface matching procedure will take $simulator$'s $rainfall$ operation as a basis to refine and rank the DCCs search. As illustrated in the first column of Fig. 4, to specify the query, to be used in the refining and ranking process, the inputs are transformed in outputs and vice-versa. The other two columns of Fig. 4 display a clip of the OWL-S descriptions of DCC1 $getQuantity$ operation and DCC3 `Weather Repository's` $getPhysicalQuantity$ operation, both retrieved in previous steps of this search procedure.

Requested Operation	Rainfall Maps of São Paulo getQuantity operation	Campinas Weather Repository getPhysicalQuantity operation
<hasInput> <Input r:ID="coordinate"> <parameterType r:resource= "&s;GeographicalCoor..."/> <g:coverage r:resource="&p;Campinas"/> </Input> **</hasInput>** **<hasOutput>** <Output r:ID="value"> <parameterType r:resource="&x;integer"/> <g:measures r:resource="&t;Rainfall"/> </Output> **</hasOutput>**	**<hasInput>** <Input r:ID="coordinate"> <parameterType r:resource= "&s;GeographicalCoor.."/> <g:coverage r:resource="&p;SaoPaulo"/> </Input> **</hasInput>** **<hasOutput>** <Output r:ID="value"> <parameterType r:resource="&x;integer"/> <g:measures r:resource="&t;Rainfall"/> </Output> **</hasOutput>**	**<hasInput>** <Input r:ID="coordinate"> <parameterType r:resource= "&s;GeographicalCoor.."/> <g:coverage r:resource="&p;Campinas"/> </Input> **</hasInput>** **<hasOutput>** <Output r:ID="value"> <parameterType r:resource="&x;integer"/> <g:measures r:resource= "&t;PhysicalQuantity"/> </Output> **</hasOutput>**

Figure 4. Clips of OWL-S specifications for a queried interface and DCC1 and DCC3 interfaces.

Using the matching procedure, the DCC1 operation is classified as plug-in. Its output description has an equivalence relationship on the query for both the $parameterType$ concept (`integer`) and the $measures$ concept (`Rainfall`). Its input description has a subsumption relationship on the `São Paulo` $coverage$ concept (since in POESIA `São Paulo` subsumes `Campinas` – see Fig. 1) and an equivalence relationship for

the $parameterType$ concept (`GeographicalCoordinates`). The DCC3 operation is classified as wider, following the same reasoning.

4 DCC-based Application Construction

The previous sections illustrated how to discover DCCs necessary to build a composition. This section shows the construction of an application that is built by composing these DCCs. Fig. 5 shows the diagram of the application. It was constructed via a composition of five DCCs, whose purpose is to simulate the growth of a coffee plant in a region of Campinas. Many aspects are omitted to simplify the example.

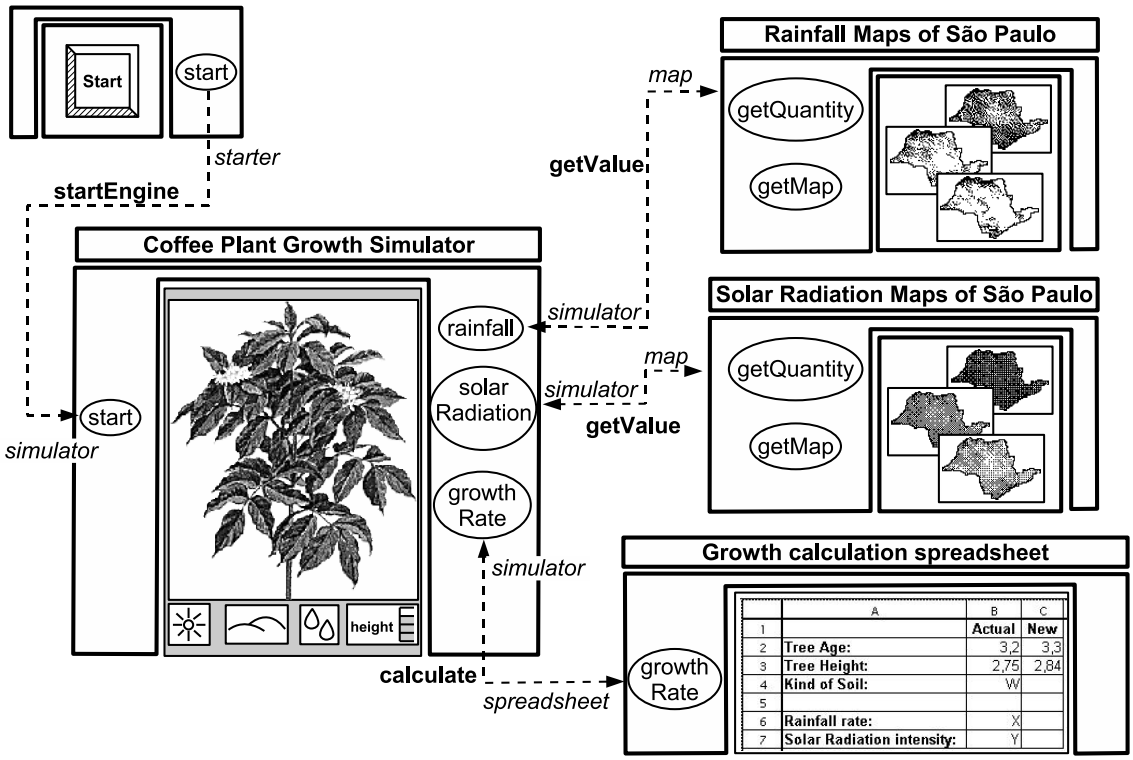

Figure 5. Composition to simulate coffee growth at São Paulo state.

Roughly speaking, an application can be constructed using the following steps: (1) elicit requirements with help of experts and users; (2) determine basic data and processes needed; (3) search for appropriate process and passive DCCs to be reused using the semantic annotations provided by DCC description in metadata and interface sections; (4) construct new DCCs if needed; (5) create the application, which is materialized into a new DCC, by appropriate composition of reused and new DCCs.

In Fig. 5, DCC interfaces show the names of operations defined via OWL-S/WSDL. The Rainfall and the Solar Radiation maps are instances of the passive DCC of Section 2.2, and the Coffee plant simulator is an instance of the process DCC of Section 2.3.

The dashed lines represent the connections between components, whose format is an adaptation and simplification of WS Choreography [Burdett and Kavantzas 2004]. To understand the purpose of these lines and their labels, we will summarize some key aspects of our model that adapt WS-Choreography concepts.

A composition is formed by a set of *participants*. Each participant is defined by a set of observable behaviors, which together form a *role* of this participant in the composition. A *relationship* is the association of two roles for a purpose. In Fig. 5 a dashed line represents a relationship between participants (DCCs), with a boldface label indicating its name. Each label in italics represents the role played by the corresponding participant in the relationship.

A given component may play several roles in a composition. In this example, each component plays a single role. Application execution starts when the user presses the start button ($starter$ component), which sends a message to the $simulator$ component. The $simulator$ graphically presents the growth of a coffee seedling in Campinas.

Before the execution of the simulation, the user configured the $simulator$ component, selecting the number of cycles in the simulation. A cycle shows the growth of a coffee plant in a time period and runs as follows. The simulator sends requests to the map component, for a period of time and region, receiving the average rainfall and solar radiation for that period and region (here, Campinas). Next, it requests that the calculator component computes the plant's state based in these and other parameters. This response is used to feed the simulator's growth rate process, showing the plant's next stage.

5 Comparison to Related Work

This section analyzes related work. We focus on two relevant aspects: our choice for describing DCC functionality; and the technique to search for DCCs.

5.1 Associating functionality to DCCs

Many initiatives identify the importance of representing some kind of relationship between the reused content and the program code, to: guarantee correct future content interpretation in long term preservation [CCSDS 2002], enable active interaction between an educational environment and units of educational content [McDonald et al. 2004], standardize the way of how multimedia content artifacts will be accessed by software units [ISO/IEC 2001], and process the content on demand to produce new transformed results [The Fedora Project team 2002].

More specifically, standards are being proposed in education [IEEE L.T.S.C. 2002, McDonald et al. 2004], digital libraries [CCSDS 2002], multimedia [ISO/IEC 2001], software development related artifacts [OMG 2004], among others. Common problems to be faced include standards to: store the content, pack and deploy autonomous reusable units and define metadata standards to describe these reusable units.

Our proposal represents a step beyond, since it provides a standard ontology-based mechanism to relate units of software and content, and provides a unified reuse perspective, suitable for both software and content, which are reused together. The tight dependency between them results in a synergetic effect that increases reuse opportunity.

5.2 Searching DCCs

There are many kinds of proposals for searching for content on the Web. We follow the trend that concentrates on using taxonomic ontologies as basis for semantic similarity.

Strategies include: the use of minimum path length between two concepts in IS-A hierarchies [Rada et al. 1989], or the maximum value of information content achieved between a concept that subsumes two compared concepts [Resnik 1995]. When the search is ontology-based, many ontologies can be involved. The ontologies related to a searched content can be different from those related to content artifacts in the repository. The issue of ontology mapping is treated in semantic integration research [Noy 2004]. If no mappings are available among ontologies, concepts related to them cannot be compared and a mapping discovery process may be needed [Doan et al. 2003].

Our proposal considers the existence of mappings between compared concepts. It is based on combining the work of Prieto-Díaz [Prieto-Díaz 1989] and Paolucci et al [Paolucci et al. 2002], using the similarity concepts proposed by Zaremski and Wing [Zaremski and Wing 1997].

Prieto-Díaz proposes the faceted method, borrowed from library science, to classify software components [Prieto-Díaz 1989]. Each facet, used to describe a component, is associated with a scheme, which defines a list of terms that can be used as facet values. To enhance component searching he uses a thesaurus to disambiguate terms and find synonyms, and a conceptual distance graph to rank similar components, based on closeness of related terms. Our approach is based on the same ideas. However, it uses OWL as a unified technology for the three tasks: properties are used as facets, taxonomic ontologies are used as thesauri, and ontological relationships are used to determine component similarity instead of conceptual distance graphs.

Zaremski and Wing [Zaremski and Wing 1997] adopt a language called Larch/ML to specify interfaces of software components and to specify queries to search required components. These specifications define pre-/postconditions for component execution using first-order predicate logic. The matching between the required component specified by a query (Q) and the provided component with interface specification (S) is based on logical relationships, like equivalence and implication. The matching between S and Q can relate pre and postconditions as separate entities, or can relate entire specification predicates S_{pred} and Q_{pred} where, for any specification X, $X_{pred} = X_{pre} \Rightarrow X_{post}$. On one hand, our approach is capable of exploring the richness of ontological relationships to compare input/output similarity, instead of logical relations. On the other hand, pre-/postconditions can detail requirements, which are not possible in our approach.

Our work is likewise related with Paolucci et al [Paolucci et al. 2002], which is also based on [Zaremski and Wing 1997], and adopts DAML-S for interface matching in Web services discovery process. As mentioned before, their approach for plug-in match follows a direction distinct from that of [Zaremski and Wing 1997]. Both approaches are contemplated in the third step of our search procedure, which is not restricted to finding software components or services, but extends this functionality-based search technique to any kind of digital content.

6 Concluding Remarks

This paper presented a new approach to structure digital content in order to facilitate its reuse and discovery using Web standards. Our work combines proposals to use interface specification, taxonomic relationships between concepts and interface matching, to enhance digital artifacts searching, using Semantic Web-based metadata and interface

specifications in our Digital Content Component model.

One of the main challenges was the specification of the functionality of each reusable piece, which guides their discovery and combination. DCC diversity requires an expressive and flexible mechanism, equally suitable for software components, images, texts, videos, among others.

We can single out two main contributions of our work in this context: first, the extension of the interface specification to express the "potential functionality" related to any kind of digital artifact, associated to a mechanism that converts it in a "real functionality" implemented by a companion component; second, a three step procedure that explores metadata associated to DCCs, combined with the functionality expressed in their interfaces, to enhance the DCC searching process.

Traditionally, the relationship between software components and Web services is related to distributed components. Here, we propose that the same technology be applied to any kind of DCC (distributed or otherwise). Semantic Web based standards are useful to promote interoperability via components, even at the local level. Therefore, we adapt these standards, simplifying them when needed to accommodate the local context. DCC description is based on an adaption of WSDL and OWL-S to describe the interface at syntactic and semantic levels respectively. This promotes reusability and facilitates the discovering of reuse units on the Web. Ongoing work involves implementation of DCC construction and search mechanisms. We have already developed a few experiments that show the feasibility of our ideas.

References

Burdett, D. and Kavantzas, N. (2004). WS Choreography Model Overview – W3C Working Draft 24 March 2004. `http://www.w3.org/TR/2004/WD-ws-chor-model-20040324/`, accessed on 12/2004.

CCSDS (2002). Reference Model for an Open Archival Information System (OAIS) – Blue Book. Technical Report CCSDS 650.0-B-1, Consultative Committee for Space Data Systems. `http://www.ccsds.org/CCSDS/documents/650x0b1.pdf`, accessed on 11/2004.

Chinnici, R. et al. (2004). Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language – W3C Working Draft 3 August 2004. `http://www.w3.org/TR/2004/WD-wsdl20-20040803/`, accessed on 11/2004.

Cullot, N., Parent, C., Spaccapietra, S., and Vangenot, C. (2003). Ontologies: A Contribution to the DL/DB debate. In *Proc. of the 1st International Workshop on the Semantic Web and Databases, 29th International Conf. on Very Large Data Bases*.

Doan, A., Madhavan, J., Dhamankar, R., Domingos, P., and Halevy, A. (2003). Learning to match ontologies on the semantic web. *The VLDB Journal*, 12(4):303–319.

Fileto, R., Liu, L., Pu, C., Assad, E. D., and Medeiros, C. B. (2003). POESIA: An ontological workflow approach for composing Web services in agriculture. *The VLDB Journal*, 12(4):352–367.

IEEE L.T.S.C. (2002). Draft Standard for Learning Object Metadata – IEEE 1484.12.1-2002. http://ltsc.ieee.org/doc/wg12/LOM_1484_12_1_v1_Final_Draft.pdf, accessed on 10/2003.

ISO/IEC (2001). ISO/IEC TR 2100-1 – Information technology – Multimedia framework (MPEG-21) – Part 1: Vision, Technologies and Strategy.

Klischewski, R. (2003). Semantic web for e-government. *Lecture Notes in Computer Science*, 2739:288–295.

Martin, D. et al. (2004). OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.1/overview/, accessed on 12/2004.

McDonald, W. A., Hyde, J., and Montgomery, A. (2004). CMI Guidelines for Interoperability AICC. http://www.aicc.org/docs/tech/cmi001v4.pdf, accessed on 11/2004.

Mørch, A. I., Stevens, G., Won, M., Klann, M., Dittrich, Y., and Wulf, V. (2004). Component-based technologies for end-user development. *Commun. ACM*, 47(9):59–62.

Noy, N. F. (2004). Semantic integration: a survey of ontology-based approaches. *SIGMOD Rec.*, 33(4):65–70.

OMG (2004). Reusable Asset Specification. http://www.omg.org/cgi-bin/doc?ptc/2004-06-06, accessed on 10/2004.

Paolucci, M., Kawamura, T., Payne, T. R., and Sycara, K. P. (2002). Semantic Matching of Web Services Capabilities. In *ISWC '02: Proc. of the First International Semantic Web Conf. on The Semantic Web*, pages 333–347. Springer-Verlag.

Prieto-Díaz, R. (1989). Classification of reusable modules. In Biggerstaff, T. J. and Perlis, A. J., editors, *Software reusability: vol. 1, concepts and models*, pages 99–123. ACM Press.

Rada, R., Mili, H., Bicknell, E., and Blettner, M. (1989). Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(1):17–30.

Raskin, R. and Pan, M. (2003). Semantic Web for Earth and Environmental Terminology (SWEET). In *Proc. of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*.

Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. In Mellish, C. S., editor, *Proc. of the Fourteenth International Joint Conf. on Artificial Intelligence*, pages 448–453.

Santanchè, A. and Medeiros, C. B. (2004). Managing Dynamic Repositories for Digital Content Components. In *Current Trends in Database Technology – EDBT 2004 Workshops*, volume LNCS 3268/2004. Springer-Verlag Heidelberg.

Smith, M. K., Welty, C., and McGuinness, D. L. (2004). OWL Web Ontology Language Guide – W3C Recommendation 10 February 2004. http://www.w3.org/TR/2004/REC-owl-guide-20040210/, accessed on 11/2004.

The Fedora Project team (2002). Mellon Fedora Technical Specification. `http://www.fedora.info/documents/master-spec-12.20.02.pdf`, accessed on 12/2004.

Zaremski, A. M. and Wing, J. M. (1997). Specification Matching of Software Components. *ACM Trans. Softw. Eng. Methodol.*, 6(4):333–369.