Applying Referent Tracking to the Use and Evolution of Websites

Werner Ceusters[1], Shahid Manzoor[1]

[1] Ontology Research Group, NYS Center of Excellence in Bioinformatics & Life Sciences
SUNY at Buffalo, 701 Ellicott street, Buffalo NY 14203, USA
ceusters@buffalo.edu
smanzoor@buffalo.edu
http://org.buffalo.edu/RTU

ABSTRACT. Many websites or parts thereof are information resources that contain descriptions about what is believed to be the case in (first-order) reality in the eyes of the website authors. When reality or the beliefs therein change, so should do the website. Interestingly, any change in a website, is an additional (second-order) change in reality itself. And so is any visit to a website. In this paper, we report on our work to develop a website that allows any such change in first- or second-order reality to be tracked. This is achieved by linking the site to a Referent Tracking System which assigns a globally unique and singular identifier to each entity in reality that is relevant to be tracked, including its own content. These identifiers are used in composite representations that describe the relationships amongst these entities following the realist agenda of Basic Formal Ontology.

1 Introduction

We are most likely moving towards a society which is such that if for something there is no representation on the Web that something probably doesn't exist. The number of websites, currently estimated to be about 180 million, is growing exponentially since the mid-nineties, and so also is the number of data-elements stored in digital resources of various kinds, many of which are also made available to authorized users over the Internet. The ultimate end of this evolution might very well be the existence of a society of information systems which contain all together a digital copy of the world which is updated synchronously with every change in the world that matters, including some changes in the information itself.

In [Ceusters & Manzoor 2009] it is described how such an evolution might be supported by the use of networks of Referent Tracking Systems (RTS). A RTS is a special kind of digital information system which keeps track of (1) what is the case in reality and (2) what is expressed in other information systems about what is believed to be the case in reality. It does so by using the Referent Tracking (RT) paradigm for information management that is distinct from other approaches in that each data element has to point to a portion of reality in a number of predefined ways. RT has been introduced in the context of Electronic Health Record keeping [Ceusters & Smith 2005, 2006b], but its

applicability is wider than that, examples being digital rights management [Ceusters & Smith 2007b] and corporate memories [Ceusters & Smith 2007a].

In this paper, we report on our work to redesign the website of the Referent Tracking Unit (RTU) within the Ontology Research Group in such a way that it satisfies the applicable principles underlying RT which themselves are based on the principles proposed in Basic Formal Ontology [Grenon 2003] as they are applied in the area of biomedicine [Smith et al. 2007] and more recently also in information management [Ruttenberg et al. 2008]. The goal of this effort is double: (1) to understand better the ontology of information artifacts and their use, and (2) to test the adequacy of RT as a tool for faithful reality representation.

2 A Realist's View on Websites

A *website* is a collection of web pages, images, videos or other digital information artifacts that is hosted on one or more web servers, and is usually made accessible via the Internet. A *web page*, in its most simple form, is an electronic document, typically written in HTML or XHTML, that is almost always accessible via HTTP, a protocol that transfers information from the web server for display in the user's web browser. Simple web pages are stored on the web server in the same form as the user will view them when *visiting* the website. In contrast to these *static* websites, there are also *dynamic* websites that generate web pages on the fly by retrieving the content from a repository of web page components. One motivation for this approach is that it is easier to maintain a few web pages with some predefined layout plus a repository that provides content, than it is to build and update hundreds of individual web pages and links. Another one is to vary the content of a page on the basis of certain criteria.

For clarity, we will use the term '*browser page*' for the copy of a web page received by the user on the basis of a request and which in most browsers is that what can be viewed when hitting the source button. For what is viewed automatically when a browser receives a browser page, we will use the term '*browser page rendering*' (or '*rendering'* for short). We reserve the term '*server page*' for that on the side of the server from which the browser page is build. Whereas the browser page is always a single digital file, the server page, in contrast, can be a collection of several files or file parts the most important ones being the *header*, *footer* and *content file*, the latter containing the *main content*. By '*main content'* we mean that part of the content of a digital file which pertains exclusively to the information that the author of the web page wanted to store in the file, thus excluding anything that is added by operating systems (such as beginning and end of file markers) or file manipulation protocols, but including for instance *layout information* and, of course, the *propositional content* as well as any other information related to the intentions of the page author.

Fig.1 is an image of the rendering of (part of) the 'referent tracking paradigm' browser page, one of the twenty pages that currently are offered through the RTU's website. This website is designed in such a way that visitors are offered the same look

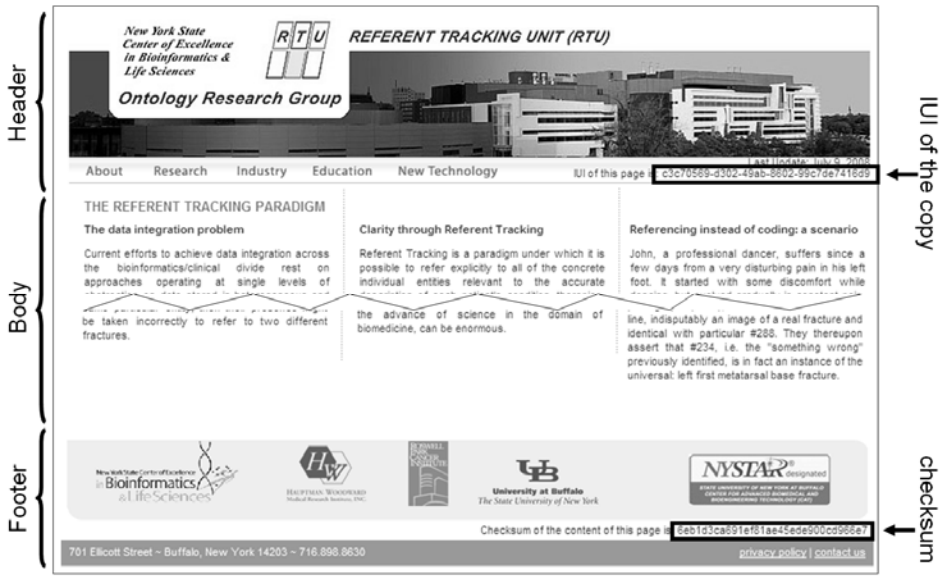

Fig.1: Layout of a browser page from the Referent Tracking Unit's website

and feel for each rendering. The header, which amongst other things contains the menu to navigate through the site, and the footer are both almost identical for each page. The body contains the main content and is distinct for each web page. This consistency is achieved by using dynamic server pages: browser pages are created dynamically by combining a header and footer file which are each *numerically the same* for all browser pages, with a content file which is distinct for each browser page.

Besides the anatomical configuration of a website, there are also two dynamic aspects that are relevant in the context of our endeavour: its use, and its evolution.

With respect to a website's use, whether or not a static or dynamic approach is taken, *that* what is on the server is from an ontological perspective never *that* what the web page visitor receives in its browser, nor *that* what he sees. In the case of a static web page, it is in fact a *copy* of the file on the server that is received, and a rendering of that copy – governed by the rendering conventions which are expressed as HTML tags – which is displayed. Two distinct users will only see a rendering of the *same* – i.e. *numerically the same or identical* – copy of a static page if they are both looking at the same screen at the same time.

Whenever a user hits the refresh button of a browser which is configured in such a way that a new request is issued to the remote server rather than to the local cache of the machine on which the browser is running, and if the page on the server has not been updated since the last request, he will obtain *another copy* of that page. The two copies received are identical in form and content, but nevertheless *distinct* copies. They each can, for instance, be saved under a different file name. Of course, in between two visits,

the page on the server might have been updated, in which case the user receives a copy of that new page, that copy being not only numerically distinct from the ones received earlier, but also distinct in content.

The request to view a page is an event of which the existence and nature is documented in the request message that is generated by the browser while executing his part of the HTTP communication protocol. Parts of this message relevant for our purposes contain information about the website host, the server page being requested, some characteristics about the requesting software agent such as type and version of the browser, the operating system under which the browser runs, the time the request was issued, the IP address of the user's machine, the latter's host name as well as the machine's port through which the communication is set up.

The second ontologically interesting dynamic aspect of a website is its evolution over time. Under the adagio that information is outdated at the time of publication, high quality websites undergo frequent changes: web pages can be added, deleted and modified, and this for a number of reasons classified in relation to that *level of reality* what undergoes the change [Ceusters & Smith, 2006a]: (1) a change in reality (level 1), for instance in case a vita page is updated when the person in question reports on the publication of a new paper or the winning of an award, (2) a change in the page author's understanding or knowledge about what is the case in reality (level 2), for instance when completing his vita page by referring to some events in the past he forgot about, or didn't assess to be important enough to be mentioned, or (3) mere changes in layout or formatting, the correction of spelling mistakes, and so forth (level 3).

3 Tracking Website Related Referents

In order to keep track of the history of a website by using a RTS, a number of principles must be applied. These principles affect both the way in which the website has to be managed, and how events that occur during the lifetime of the website are to be represented. Most important is that the entities to be tracked are denoted by an Instance Unique Identifier (IUI), the assignment of which is managed by the RTS itself. Relevant website related entities that receive an IUI in our current setup are:

- each individual HTML-encoded *content file* on the server which is made accessible for viewing as part of the RTU website (thus not the server page itself, nor the content of it, pdf-files, and so forth – see section 5 for some issues),
- the middleware component (i.e. a collection of executable software code) on the server that, when activated, manages the assignment of IUIs and the generation of representations, called '*RT-tuples*', in the RTS (a list of templates for RT-tuples is shown in Table 1),
- each individual browser page that is requested, irrespective of whether the corresponding server page has changed. The IUI for a browser page from the RTU website is displayed right of the menu bar (Fig.1),
- each request message.

This list will very likely increase in size in the near future.

In addition, to satisfy the RT principles [Ceusters 2007, Ceusters et al. 2006, Ceusters & Smith 2006b], there must be an IUI for each ontology and terminology that is used to describe the entities that are tracked.

Whereas new *browser pages* receive automatically an IUI, an IUI for the content of *server pages* is generated on demand of the page author only, typically, but not always or automatically, whenever the server page is updated. This is because one of the principles of RT is that only entities in reality which are *relevant* for some purpose should receive an IUI. Because of the dynamic aspect of the RTU website, server pages are only accessible to the site's authors and administrators and are thus only relevant to them (at least in the current setup).

Table 1: RT-tuple templates registered in a Referent Tracking System

Tuple name ***Attributes*** Description
A-tuple ***< IUI_a, IUI_p, t_{ap}>*** Act of assignment of *IUIp* to a particular at time t_{ap} by the particular referred to by author IUI_a
D-tuple ***< IUI_d, IUI_A, t_d, E, C, S >*** A D-tuple is inserted (1) to resolve mistakes in RTS, and (2) whenever a new tuple other than a D-tuple is inserted in the RTS. The particular referred to by IUI_d registers the particular referred to by IUI_A (the IUI for the corresponding A-tuple) at time t_d. E is either the symbol 'I' (for insertion) or any of the error type symbols as defined in (Ceusters, 2007). C is the reason for inserting the A-tuple. S is a list of IUIs denoting the tuples, if any, that replace the retired one.
PtoP-tuple ***<IUI_a, t_a, r, IUI_o, P, t_r>*** The particular referred to by IUI_a asserts at time t_a that the relationship *r* from ontology IUI_o obtains between the particulars referred to in the set of *IUIs P* at time t_r.
PtoU-tuple ***<IUI_a, t_a, inst, IUI_o, IUI_p, UUI, t_r>*** The particular referred to by author IUI_a asserts at time t_a that the particular referred to by IUI_p instantiates – by means of the *inst* relation defined in ontology IUI_o – the universal *UUI* at time t_r.
PtoC-tuple ***<IUI_a, t_a, IUI_c, IUI_p, CUI, t_r>*** The particular referred to by IUI_a asserts at time t_a that at time t_r concept code *CUI* from terminology system IUI_c is an accurate term for IUI_p
PtoU(-) -tuple ***<IUI_a, t_a, r, IUI_o, IUI_p, UUI, t_r>*** The particular referred to by IUI_a asserts at time t_a that the relation *r* of ontology IUI_o does not obtain at time t_r between the particular referred to by IUI_p and any of the instances of the universal denoted by *UUI* at time t_r.
PtoN ***< IUI_a, t_a, nt_j, n_i, IUI_p, t_r, IUI_c>*** The particular referred to by IUI_a asserts at time t_a that n_i is the name of the

nametype nt_j used by IUI_c to denote the particular referred to by IUI_p at t_r.

One good reason, so we believe, for assigning a new IUI to an updated content file is version management and this from a double perspective: it allows to keep track of how server pages evolve over time and what browser pages have been generated thereof and then transmitted to users. Keeping track of this is an absolute requirement for the digital copy of the world mentioned in the introduction not just to be a faithful copy of what is the case *now*, but also of the world's *history*.

An extra feature of the RTU website is that each browser page contains a *checksum* which is calculated over the main content and the IUI of the browser page. The purpose of this checksum is to detect modifications to the page and to serve as a safeguard for both website users and authors. In case a user forges a browser page in an attempt to claim that certain content was present while it actually was not, recalculation of the checksum will make the fraud evident. On the flip side, the website authors can never deny that certain content *was* there, because the recalculation of the checksum on an unaltered browser page will return the same result. Browser pages created from the same server page differ only in both IUI and checksum.

3.1 An Example: Tracking a Request to View a Web Page

Imagine a scenario under which a visitor of the RTU website requests a copy of the server page associated with a content file to which is assigned IUI #12. Table 2 shows the RT-tuples, following the notation explained in Table 1, that are generated as a result of this request.

As soon as the request message arrives at the RTU website, the middleware component – to which is assigned IUI #2 and which functions as 'author' of RT-compatible statements – asks the RTS to generate a new IUI which will be assigned to the request message. There is no need here to verify whether this message has already been given an IUI because that is impossible: each such message is unique. As a result, the RTS inserts A-tuple #25 as an RT-compatible representation for the assignment of IUI #24 to that message. Note that the RTS itself does not know to what precisely #24 is assigned; that information is kept in a separate database of the application in which the header of the message is explicitly linked to #24. In line with the RT principles, any insertion of an A-tuple requires the insertion of a D-tuple, which in this case is the tuple to which #26 is assigned. The E-attribute of #26 being set to 'I' and the C-attribute to 'CE' indicate that the reason for the insertion is a change in reality.

Next the middleware requests the RTS to generate a new IUI which will be assigned to the browser page that will be built on the basis of #12, the requested web page. This results in the generation of #27 for the browser page and the insertion of A-tuple #28 and corresponding D-tuple #29.

As a third step, the middleware informs the RTS that #27 is the browser page generated out of #12. It does this by linking the former to the latter by means of the ***MainContentCopyOf*** relationship as defined in the Website Tracking Ontology designed for this purpose and to which is assigned IUI #022. This leads to the generation of PtoP-

tuple #30 and corresponding D-tuple #31. These two tuples demonstrate nicely how the RTS, guided by the right information from the middleware component, keeps track of the different time instants involved: it is 'known' in the RTS since time-23 that at time-22 the middleware component asserted that #27 is a content copy of #12 since time-20, the latter being the time that the assignment of #27 was requested.

Next, it is asserted that #27 is generated following the request obtained through message #24 and this by means of #022's ***InstigatorOf*** relationship (tuples #32 and #33).

The fifth action taken by the middleware component in its communication with the RTS relates to the generation of the checksum which leads to the creation of the tuples with keys #35, #36, #37, and #38. Most pertinent is the PtoP-tuple that describes that #34 denotes the checksum calculated for #27, suing #022's ***ChecksumOf*** relation.

4 Technical Implementation

We selected the web server programming language PHP as the main implementation language together with the Zend Framework, a third party component for PHP, which provides modules for building websites based on the Model View Controller (MVC) paradigm (Zend Technologies Ltd, 2009). MVC hinges on a clean separation of software engineering objects into one of three categories: models for maintaining data, views for displaying all or a portion of the data, and controllers for handling events that affect the model or views (Fowler, 2002). When a browser requests a web page from a web server, the Zend Framework (running on the web server) first executes the programming logic in its controller part with the goal to retrieve data, which is followed by the execution of the programming logic in the view part which leads to the generation of HTML using the data received earlier. This technology provides ease in managing the programming logic of a website especially when changes are made frequently. Making changes in the layout or design of a website does not always require changing the controller or model part.

4.1 Architecture

Figure 1 depicts how the various components interoperate when a user accesses the RT website's homepage via an HTTP client such as Internet Explorer. This access causes the index.php file on the server to be executed which results in an activation of the Zend Framework (ZF). ZF executes the request in several steps which includes composing the webpage such as it will be viewed by the user and activating the middleware component which communicates with the RTS to assign IUIs to the various particulars involved in the user request such as the http header representing the user request, the requested web page, and so forth.

We needed to implement only one controller (called the *IndexController* in line with the naming conventions on which the Zend Framework relies) and one action (*indexAction*). This indexAction is responsible to process all the requests of the RT website and

Table 2: RT-tuples registered in the RTS upon the request to send a browser page. The column labelled n shows the order in which the tuples are inserted. The column labeled Key contains the IUI for that tuple. The other columns contain the attributes specific for each RT-tuple template as specified in Table 1. The descriptions in the time-related columns tap, td, ta and tr follow [European Committee for Standardization 2005].

A-tuples				
n	**IUI_p**	**IUI_a**	**t_{ap}**	**Key**
1	#24	#2	(EVENT("#24 assignment") has-occ AT TP(time-18))	#25
3	#27	#2	(EVENT("#27 assignment") has-occ AT TP(time-20))	#28
9	#34	#2	(EVENT("#34 assignment") has-occ AT TP(time-26))	#35

D-tuples							
n	**IUI_d**	**IUI_A**	**t_d**	**E**	**C**	**S**	**Key**
2	#2	#25	(EVENT("#25 inserted") has-occ AT TP(time-19))	I	CE		#26
4	#2	#28	(EVENT("#28 inserted") has-occ AT TP(time-21))	I	CE		#29
6	#2	#30	(EVENT("#30 inserted") has-occ AT TP(time-23))	I	CE		#31
8	#2	#32	(EVENT("#32 inserted") has-occ AT TP(time-25))	I	CE		#33
10	#2	#35	(EVENT("#35 inserted") has-occ AT TP(time-27))	I	CE		#36
12	#2	#37	(EVENT("#37 inserted") has-occ AT TP(time-29))	I	CE		#38

PtoP-tuples							
n	**IUI_a**	**t_a**	**r**	**IUI_o**	**P**	**t_r**	**Key**
5	#2	(EVENT("#30 is asserted") has-occ AT TP(time-22))	MainContent-CopyOf	#022	#27, #12	(EPISODE("#30 is true") has-occ SINCE TI(time-20))	#30
7	#2	(EVENT("#32 is asserted") has-occ AT TP(time-24))	InstigatorOf	#022	#24, #27	(EVENT ("#32 is true") has-occ AT TP(time-18))	#32
11	#2	(EVENT("#37 is asserted") has-occ AT TP(time-28))	ChecksumOf	#022	#34, #27	(EPISODE("#37 is true") has-occ SINCE TI(time-26))	#37

is invoked automatically by a preset mapping of the website's URL to this component.

Rather than defining separate actions for each individual webpage out of which our website is built, we used the HTTP *parameter-value mechanism* to process requests for specific web pages such as the '*Papers*' or '*Presentations*' web pages. The same indexAction retrieves the user request page parameter (e,.g. ?page=presentations) and sets the path of the requested page which is later used by the index.phtml file to load the file for rendering.

The layout of our website is pre-programmed in the main.phtml file which contains a space in which the contents of the requested web page are copied together with the IUI obtained through the communication with the RTS.

The content files are stored as PHP documents with a phtml extension but are encoded in pure HTML without any PHP code. All content files are stored on the server in a designated path based on the filename and a version number. This setup allows the page owner to manage versions in a detailed way: depending on what he changes pre-

cisely, he can decide to consider the content file to be new or the same (though of course changed), and in the latter case to be of a new version or not. Thus files and contents, as continuants under the realist agenda, endure over time while undergoing changes or give rise to new files.

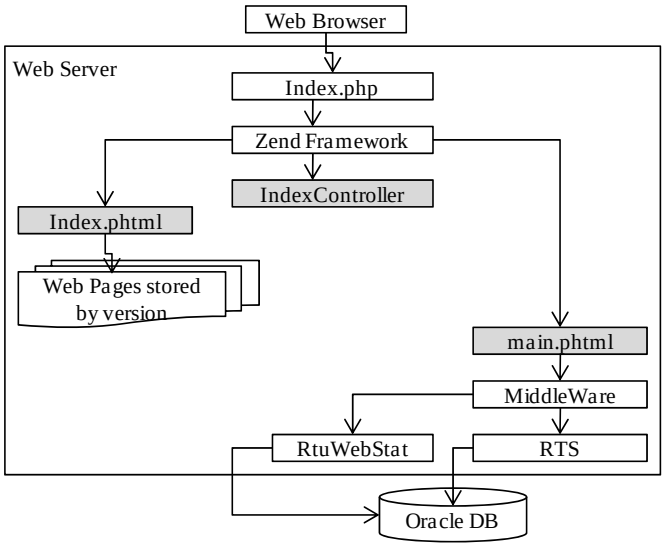

Fig.1: RTU Website implementation Architecture

4.2 Steering the RTS

The middleware component is informed about changes in a server page by means of the HTML 'input' tag inside the corresponding content file. This tag is coded with the type value 'hidden' which instructs the browser not to render its content on the screen. This tag carries the following application specific attributes:

- ***currentIUI***: contains the IUI for the content file and is always set by the middleware component;
- ***previousIUI***: is set by the page author and contains the IUI of the content file from which the content of the current page is derived;
- ***newNameForPreviousFile***: is set by the page author to be the new name of the content file from which the current web page is derived. This attribute is part of the versioning mechanism as it allows to retain an older version under a new name. An alternative would have been to let the older version keep its name, and to assign a new name to a new version, but this would require to use (possibly very long) HTML-encoded redirection chains to en-

able links stored externally (for instance in a visitor's favorite links collection) to lead automatically to the newest version.

- ***previousVersion***: is set by the page author to contain the version number of the content file from which the current content page is derived and judged to be a new version.

When a web page is created for the first time, these attributes are left empty.

It is the middleware component that does all the processing related to the assignment of IUIs to the particulars involved in a web request after receiving the contents of the requested web page from the layout view component of the Zend Framework.

When processing the server page, the middleware component checks whether the requested web page needs to be assigned an IUI. Its decision is based on the value set for the currentIUI attribute: when empty, the server page has to be assigned an IUI. In case this attribute has already a value, the component recalculates by means of the md5-algorithm an 128-bit checksum and verifies whether it matches the checksum that was assigned to it when the file was created. Any difference indicates a modification to the contents of the file. The middleware component is set up in such a way that this event triggers the assignment of a new IUI. In this case it sets (1) the value of the previousIUI attribute as the value of currentIUI attribute and (2) the value of currentIUI empty. Then starts a communication session with the RTS to perform the following steps:

1. it requests the RTS to generate a new IUI for the modified content;
2. it puts that IUI in the currentIUI attribute of the input tag,
3. it starts the checksum-related processing for the web page which consists of:

    a. calculating the checksum over the content,
    b. adding the checksum as a hidden variable to the server page,
    c. requesting the RTS to generate an IUI for the checksum,
    d. storing the checksum and the corresponding IUI in a database table,
    e. instructing the RTS to insert a PtoP-tuple representing that the ***ChecksumOf*** relation holds between the checksum and the server page,

4. it requests the RTS to insert a PtoN-template indicating the name of the content file,
5. if previousIUI is not empty but previousVersion is empty, it requests the RTS to insert a PtoP-tuple stating that the ***DerivesFrom*** relation [Smith et al. 2005] holds between the current and previous content,
6. if both previousIUI and previousVersion are not empty, it requests the RTS to insert a PtoP-tuple stating that the ***newVersionOf*** relation holds between the current and previous content,
7. it saves the content file on the web server.

Generating the browser page is achieved by performing the steps described in detail in section 3.1.

5 Discussion

The Referent Tracking enabled RTU website went life in September 2008 but is nevertheless still work in progress; several computational and philosophical challenges, some not anticipated, are still not fully addressed.

5.1 Computational issues

From a computational perspective, there is the massive growth of the RTS database: in its current implementation, one visitor request leads to the generation of 12 RT-tuples. An update of a server page leads to 15 new RT-tuples which includes the reformulation of existing tuples based on the changes in reality that have occurred. It is very likely that when our insight in the ontology of information artifacts grows, as well as in the various ways in which pieces of information, depending on how and what they represent, age over time, so also will increase the number of RT-tuples that need to be generated.

As an example, when by means of a PtoN-tuple it is asserted that a name is assigned to an entity, then it is specified that the name is applicable ***since*** the datetime the name was assigned. If in the RTS that entity was already stated to have a name associated with it, then that earlier name assignment was of course also done by means of a similar PtoN-tuple for the previous name of that entity. But when the new name is assigned, that already existing PtoN-tuple does not correspond with reality anymore. Because one of the principles of RT is to generate a faithful representation of reality throughout reality's entire (registered) history, that tuple must be 'retired', where 'retired' means: that what is described by the tuple was true at that point in time, but is not true anymore. This is achieved by adding an appropriate D-tuple to the RTS [Ceusters, 2007]. A new PtoN-tuple, still about the previous name, will then be generated to represent that the previous name was valid from the time that name was assigned, ***until*** the time the new one was assigned.

A computational problem we had not foreseen was the behaviour of some crawlers after visiting our website. Some crawlers, including alert generators, keep track of the frequency with which pages are updated: the more frequent a page is updated, the sooner they revisit the website. Being 'updated' for such crawlers means: the file found at the URL during the latest visit is not exactly the same as the one found during the previous visit. But since crawlers, as any other visitor to the RTU website, receive browser pages, they ***never*** receive a file which is identical to any previous file because browser pages carry a unique identifier which is each time distinct from any other identifier. As a result, we detected at least one crawler that spent days on visiting – and revisiting – our site. This might seem to be attractive if one is interested in generating

frequent visits to the site, but at the other hand, it overloads the RTS with tuples that are not really relevant, thereby – perhaps – violating one of the principles of RT itself.

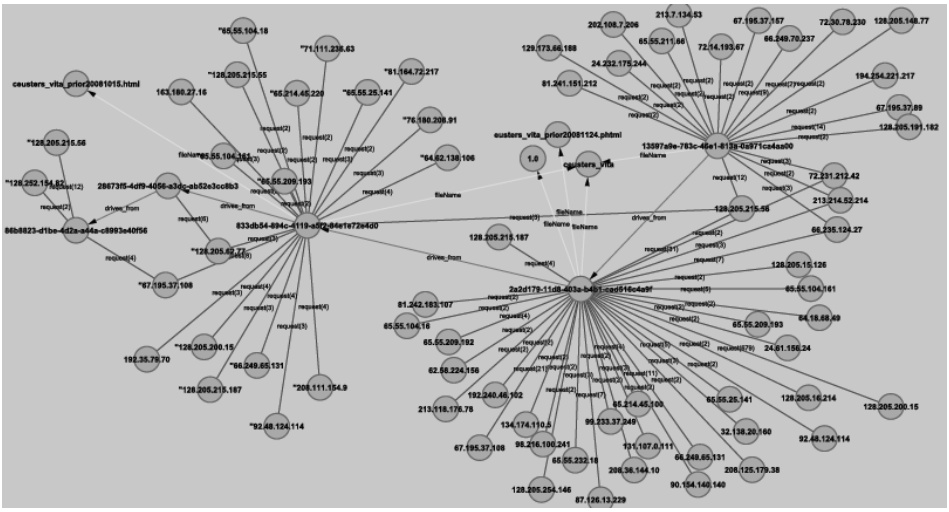

Fig.2: Network view of (part of) the RTU website's RTS showing how the content of one page changed over time and which IP addresses received a browser page generated from these contents

Another challenge, well known in computer science circles, is how to visualize the results in a user-friendly way. Figure 2 shows a typical network graph, in this case concerning the changes over time of the content and name of content pages, as well as the IP addresses from which requests concerning that page were issued. IP addresses from which our site seems to be monitored can be spotted easily as they have connections to more than one content node (i.e. the nodes which a large number of links). But visualizing in this way the entire history of a site, will lead to the well-known *hairball* problem.

5.2 Ontological issues

The goal of the project forces us to have a good understanding of information artifacts, and in particular digital information artifacts. It is thus no surprise that the project runs in parallel with the Information Artifact Ontology initiative that has recently been set up but thus far is mainly focused on the data representation needs in the context of clinical trials (Ruttenberg et al. 2008).

One issue is the lack of clear ontological criteria for what constitutes a change in content versus new content. Another one is how this relates to the coming into being of a new content file versus the modification of an existing one. A third issue is how to characterise and define the distinct sorts of content that need to be differentiated.

Imagine that a researcher maintains his vita on his laptop and that this document has the name 'my-vita'. If registered in a RTS, that document would be assigned an IUI, and possibly also its content. When the researcher wants to add to it, he opens that document, adds a few lines, saves it and then closes it. In this case, we would argue that no new document is involved, but that the document in question underwent a change, and so its content. Thus in terms of the OBO Relation Ontology [Smith et al. 2005] there would be two ***transformations***. If however the researcher saves the result of his editing as a new file, then that file would be some sort of ***derivation*** from the other file. But what about the content? Because saving the document as a new one does by itself not change anything about the content, one could argue that the 'new' content is also a transformation of the 'old' content, which is saying that it is still the *same* content that underwent a change.

Now imagine that the researcher copies the 'my-vita' file, assigns the name 'my-own-vita' to it, and then opens in his word processor the 'my-vita' file with the goal to replace the sentences describing his scientific life with those describing his fellow researcher's scientific life. He does this because his friend is not good in dealing with computers. Moreover, because he invested already a lot of time in setting up a nice layout for his own vita, it is easier to maintain this layout by pasting 'new' information over the existing one, than to build his friend's vita from scratch. He also keeps the filename the way it is, because, when delivered to his friend, the name 'my-vita' will make perfectly sense to that friend.

The actions performed in editing under this new scenario the same document as before, are not different than under the first scenario. However, it is this time not so clear that there are arguments to say that a transformation is involved: perhaps for the file (as long as we are talking about the file on the laptop, and not a copy of it on another carrier), but for sure not for the content. So where are the boundaries to be drawn? It is only by answering these questions that we will be able to give a formal definition for relations such as ***MainContenCopyOf*** and ***DerivesFrom*** in the context of digital information artifacts rather than the intuitive use we currently make of them.

Another issue to be dealt with is an appropriate treatment of time. A good basis is offered by [European Committee for Standardization 2005] which is a standard for describing states of affairs in relation to time. The central notion is that of 'situation' which is defined as a '*phenomenon occurring (or having the potential to occur) at or over a time in a given world context*'. The representation of a situation in a language is called a '*predication'*. That part of a predication that is free from a temporal expression is called a '*propositional clause*'. Thus in the predication 'IUI #22 was assigned July 4, 2008' the part 'IUI #22 was assigned' is a propositional clause.

The standard proposes a semi-formal syntax which forces the author of a propositional clause to be explicit about whether the clause denotes a situation considered to occupy a time interval (therefore labeled 'EPISODE'), a situation considered to occur at a time point (therefore labeled 'EVENT'), or whether he is agnostic about it ('SIT'). These labels are used in the temporal attributes of the RT-tuples (see Table 2) whose values conform to the standard completely. The expressions – still following the provi-

sions of the standard – contain also 'temporal comparators' to express temporal relationships which are inspired by Allen's calculus [Allen 1984].

However, the standard does not give indications on what from an ontological perspective makes sense to express. An illustrative example of the sort of problems one can face in this context is offered by PtoP-tuple #32 (Table 2). In this tuple it is asserted that the relationship ***InstigatorOf*** holds between request message #24 and browser page #27 at time-18, i.e. the time the message was received by the middleware component. But at that time, the browser page did not yet exist. Thus if A causes B – ***InstigatorOf*** is a sort of causal relationship – at what time would the causal relationship then hold? Addressing these issues is part of the work that needs to be carried out, not necessarily just by us, but by the ontology community in general.

6 Conclusion

By implementing a Referent Tracking enabled website, we have been able to give a concrete example of the benefits that the Referent Tracking paradigm has to offer. The use of unique identifiers combined with checksums for verifying the content-integrity of web pages holds promises for a safer and more reliable web. Our effort also demonstrates the feasibility of working with IUIs in an almost completely autonomous way, thus providing an answer to the question often brought up by RT sceptics: 'who has the time to request IUIs and to specify all these relationships?' As we have demonstrated earlier in the context of linking electronic health records to a RTS as well [Manzoor et al. 2007; Rudnicki et al. 2007], many assignments and representations can be generated automatically. More work is still to be done, and many questions are left open. Answering these questions will not only be valuable for our own endeavour, but also for ontological research in the domain of information artifacts.

References

[Allen 1984] Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence, 23,* 132-154.

[Ceusters 2007] Ceusters, W. (2007). Dealing with Mistakes in a Referent Tracking System. In H. KS (Ed.), *Proceedings of Ontology for the Intelligence Community 2007 (OIC-2007)* (pp. 5-8). Columbia MA.

[Ceusters et al. 2006] Ceusters, W., Elkin, P., & Smith, B. (2006). Referent Tracking: The Problem of Negative Findings. In A. Hasman, R. Haux, J. v. d. Lei, E. D. Clercq & F. Roger-France (Eds.), *Studies in Health Technology and Informatics. Ubiquity: Technologies for Better Health in Aging Societies - Proceedings of MIE2006* (Vol. 124, pp. 741-746). Amsterdam: IOS Press.

[Ceusters & Manzoor 2009] Ceusters, W., & Manzoor, S. (2009 (in press)). How to track Absolutely Everything? In L. Obrst, W. Ceusters & T. Janssen (Eds.), *Ontologies for Intelligence*. Amsterdam: IOS Press.

[Ceusters & Smith 2005] Ceusters, W., & Smith, B. (2005). Referent Tracking in Electronic Healthcare Records. In R. Engelbrecht, A. Geissbuhler, C. Lovis & G. Mihalas (Eds.), *Connecting Medical Informatics and Bio-Informatics. Medical Informatics Europe 2005* (pp. 71-76). Amsterdam: IOS Press.

[Ceusters & Smith 2006a] Ceusters, W., & Smith, B. (2006). A Realism-Based Approach to the Evolution of Biomedical Ontologies. In *Proceedings of AMIA 2006* (pp. 121-125).

[Ceusters & Smith 2006b] Ceusters, W., & Smith, B. (2006). Strategies for Referent Tracking in Electronic Health Records. *Journal of Biomedical Informatics, 39*(3), 362-378.

[Ceusters & Smith 2007a] Ceusters, W., & Smith, B. (2007a). Referent Tracking for Corporate Memories. In P. Rittgen (Ed.), *Handbook of Ontologies for Business Interaction* (pp. 34-46). New York and London: Idea Group Publishing.

[Ceusters & Smith 2007b] Ceusters, W., & Smith, B. (2007b). Referent Tracking for Digital Rights Management. *International Journal of Metadata, Semantics and Ontologies, 2*(1), 45-53.

[European Committee for Standardization 2005] European Committee for Standardization. (2005). EN 12388:2005. Health informatics - Time standards for healthcare specific problems.

[Fowler 2002] Fowler, M. (2002). *Patterns of Enterprise Application Architecture*: Addison-Wesley Professional.

[Grenon 2003] Grenon, P. (2003). *Nuts in BFO's Nutshell: Revisions to the Bi-categorial Axiomatization of BFO* (Technical Report). IFOMIS Reports: Institute for Formal Ontology and Medical Information Science.

[Manzoor et al. 2007] Manzoor, S., Ceusters, W., & Rudnicki, R. (2007). A Middleware Approach to Integrate Referent Tracking in EHR Systems. In Teich JM, Suermondt J & H. C (Eds.), *Proceedings of the American Medical Informatics Association 2007 Annual Symposium. Biomedical and Health Informatics: From Foundations to Applications to Policy* (pp. 503-507). Chicago IL.

[Rudnicki et al. 2007] Rudnicki, R., Ceusters, W., Manzoor, S., & Smith, B. (2007). What Particulars are Referred to in EHR Data? A Case Study in Integrating Referent Tracking into an Electronic Health Record Application. In Teich JM, Suermondt J & H. C (Eds.), *American Medical Informatics Association 2007 Annual Symposium Proceedings, Biomedical and Health Informatics: From Foundations to Applications to Policy* (pp. 630-634). Chicago, IL.

[Ruttenberg et al. 2008] Ruttenberg, A., Smith, B., & Ceusters, W. (2008). Information-Artifact-Ontology. Retrieved December 10, 2008, from http://code.google.com/p/information-artifact-ontology/

[Smith et al. 2005] Smith, B., Ceusters, W., Klagges, B., Köhler, J., Kumar, A., Lomax, J., et al. (2005). Relations in biomedical ontologies. *Genome Biology, 6*(5), R46.

[Smith et al. 2007] Smith, B., Ashburner, M., Ceusters, W., Goldberg, L., Mungall, C., Shah, N., et al. (2007). The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology, 25*, 1251-1255.

[Zend 2009] Zend Technologies Ltd. (2009). Zend Framework: The Official Programmer's Reference Guide: Apress.