A linked-data based virtual repository for disaster management tools and applications

Y. Z. Ou[1], S. H. Tsai[1], Y. A. Lai[1], J. Su[1], C. W. Yu[1], C. T. Hsiao[1], E. T.-H. Chu[2], K. J. Lin[3], J. M. Ho[1] and J. W. S. Liu[1]
[1]*Institute of Information Science, Academia Sinica, Taiwan*
[2]*Department of Computer Science and Information Engineering, National Yunlin University of Science and Technology, Taiwan*
[3]*Department of Electrical Engineering and Computer Science, University of California, USA*

Abstract

Nowadays, developed regions in the world, including Taiwan, have a wealth of data and information, which if made available in time for disaster preparedness and response purposes, can help save lives and minimize damage. With a few exceptions, however, state-of-the-art data management information systems (DMIS) available in most countries do not provide adequate support for search, discovery, access and use of data and information residing in independent sources across institutional boundaries. This paper describes architecture and design of a distributed middleware-level framework, called Virtual Repository (VR), together with functionalities and structures of its key components. By leveraging linked data and related technologies and tools, VR aims to eliminate this limitation. Two applications, Mobile Assistant for Disasters (MAD) and Automatic Disaster Alert System for Tourists (ADAST) are described to demonstrate the feasibility and effectiveness of VR

Keywords: linked data, disaster preparedness and response, information system.

1 Introduction

In recent years, news, historical records, published statistics and research studies on past disasters worldwide consistently tell us that data and information needed to support disaster preparedness and response decisions and operations are critical

WIT Transactions on The Built Environment, Vol 133, © 2013 WIT Press
www.witpress.com, ISSN 1743-3509 (on-line)
doi:10.2495/DMAN130171

to our abilities to cope with natural disasters, especially unforeseen catastrophic calamities. Such information, if made available in time, not only can help save lives and reduce damages for people affected by disasters, but also can make emergency response and rescue operations safer and more efficient. Statistics also show, however, that the positive impacts of information deteriorate rapidly with time following a disaster [1]. Therefore, making decision support information easily discoverable and accessible by Emergency Operation Center (EOC), emergency responders, victims and general public should be a primary design objective of all disaster management information systems (DMIS)

Today, this objective is met only partially by DMIS of numerous countries and regions in the world, including Taiwan and most of Asia. Typical state-of-the-art DMIS rely mainly on data and information in sources owned by government agencies responsible for disaster management. As a part of standard operating procedures (SOP) in preparation for a disaster, likely emergency scenarios are developed based on prior knowledge on similar disasters and experiences in dealing with them. When the disaster become imminent, the EOC determines the data and information needed to deal with the scenarios and has the data retrieved from available sources and cached on devices, computers and display systems and thus makes the data ready for use by decision makers and responders during the emergency. This practice and the DMIS used to support the SOP work sufficiently well for typhoons, downpours, earthquakes of usual severities, and other types of emergencies that occur frequently in the region.

The limitations of current DMIS have become evident time and again in unforeseen situations, however. In recent years, technologically advanced regions have a wealth of data beyond what are available in sources contained in the official DMIS. Some of these data (e.g., structures and surveillance data of damaged buildings in affected area, locations of people needed help to evacuate, real-time data on available private and public owned transports, and so on) can be invaluable when the levels of threat and devastations exceed the prediction. Yet, typical DMIS offer little or no support to enable timely discovery, access and use of such data, especially when the data are in sources outside of the official DMIS and across institutional boundaries.

This fact has motivated academics and industry, as well as governments of many countries, to exploit linked data and related technologies [2, 3] for disaster management. Adding semantics and relations to transform raw data into Linked Data (LD) not only eases the discovery and use of critically needed data during emergencies, but also enables the design and implementation of new and more effective disaster preparedness and response applications. Research projects on building emergency information system and management infrastructures on linked data and Linked Open Data (LOD) include the ones described in [4–6]. Tools provide by projects such as LOD2 [7] and SMILE [8] can help to reduce the effort and speed up the development of linked-data enabled DMIS and disaster management applications.

Building on this momentum, we proposed in a previous paper [9] architecture and design of a middleware-level framework called *virtual repository* (VR) for LD-based disaster preparedness and emergency response applications. Despite

their commonly acknowledged advantages, adoption of LD and LOD within disaster management IT infrastructures has been slow. A reason is the enormous amounts of resources and efforts required to enhance existing DMIS with semantics and links [10] in anticipation of future needs. The virtual repository framework addresses this concern by offering applications with similar data requirements an extensible repository in which Universal Resource Identifiers (URIs) and links can be created from existing data sets on demand as needs for them arises. (When there is no possibility of ambiguity, we also call such a repository a VR.) In addition to supporting run-time access of linked data, each VR comes with tools using which the developers of the applications supported by the VR can easily create new linked data and maintain existing ones as the applications and data sources served by the repository evolve.

We describe in this paper the functionalities, structures and implementations of key components of a typical VR. Following this introduction, Section 2 further motivates the VR framework and its design rationales with the help of two applications, Mobile Assistant for Disasters (MAD) and Automatic Disaster Alert System for Tourists (ADAST). These applications share a VR. They were presented in [9] as case studies. Section 3 describes the VR architecture and implementation from application development perspective. Section 4 describes in detail VR components. Section 5 discusses our future work.

2 Motivation and rationales

Again, we use MAD and ADAST to help us explain the reasons for using LD, LOD and VR framework. We also use them to explain the functionalities required of the VR to support the development and use of these and similar applications.

2.1 To support development of work-anywhere applications

MAD is an application system designed to help the general public access and download to mobile devices information on where to go and what to do when a major disaster strikes so that they can carry the information with them during the emergency. Such information includes nearby buildings designated as public shelters, parks with portable water and food, emergency medical care centers, and so on. Many city and township governments in the world (including Taipei, San Francisco, and London [11–13]) have made this kind of information available on-line. So, it should be straightforward for applications such as MAD to retrieve the information using the open interfaces and API functions of the local web services and databases. Indeed, this is what the alpha version of MAD does.

Specifically, the proof-of-concept prototype MAD v0.5 serves only people in Taipei city. The application system has a client-server structure. As its name indicates, MAD clients are applications running on mobile devices commonly used by general public every day. One (or more) interface server (IS) retrieves from Taipei OpenData [12] disaster preparedness information for general public in the city, periodically during normal times and upon notification by alert authorities. The IS then partitions the data into location-specific subsets, each for

a district of the city, and distributes the subsets pervasively to point-of-service (POS) servers running on computers and devices contributed by businesses (e.g., convenience stores and cafes) and organizations (e.g., schools and companies) within the districts. In this way, MAD makes the critically needed data highly available and downloadable to mobile clients via local connections when Internet and phone connections are disrupted.

MAD v0.5 allowed us to demonstrate the functionalities and usability of MAD but falls short on the maintainability and extensibility. A reason arises from the fact that information consumers targeted by sources such as Taipei OpenData are people, not programs. Often, inconsistencies in naming schemes and data schemas cross different domains have not been carefully eliminated. As an example, in Taipei OpenData, "property name" of a sports center and "organization name" of a hospital both refer to names of the respective facilities. Human can easily process this information despite the difference, but not programs. We solved this problem in MAD v0.5 by using brute-force, hand-coded mappings. As expected, MAD v0.5 is hard to maintain as these mappings must be updated when schemas of the data source(s) change.

We want future versions of MAD to be a work-anywhere service. In particular, the client component of MAD, once installed on a mobile device, can access and download local disaster preparedness information wherever such information and MAD service are available. Our current goal is to demonstrate that MAD 1.0 work in representative cities including Taipei, London and Tokyo. A way to accomplish this goal is to have MAD servers and clients work with a common data model and format regardless the data models, schemas and views of available local information sources. Resource Description Framework (RDF) [10] is a natural choice for this purpose: Translation tools such as the ones provided by LOD2 [7] can ease the effort of translating the structures and semantics of legacy data schemas into a RDF format. With data objects and resources named by their URIs, the problem due to multiple aliases for the same object or different objects with the same or similar names is eliminated.

An important advantage of the RDF model and associated formats over other choices of common data models, schemas and formats is that RDF enables us to link not only data sets retrieved by MAD from sources used by the system but also with data sets from external sources that support linked data. In places where local sources have linked open data (e.g., in numerous cities in the US and EU), MAD can easily discover and retrieve disaster preparedness data in a RDF format from the sources and directly forward the data to its mobile clients. At locations where the local sources do not have linked data (e.g., in Taipei) and may not even have open data (e.g., in many cities in Asia), MAD IS uses API functions, web crawlers and translation tools to retrieve the required disaster preparedness data, translate and store the data in RDF format. We advocate here that the linked data thus generated by MAD and similar applications be kept in a common triple store provided by a middleware so that the applications can share the data and the effort spent to create the links is amortized. In addition to space in the triple store, the middleware also provides supporting tools for the generation and use of linked

data in the triple store, sources with linked data and legacy sources without linked data. This middleware is what we call a virtual repository (VR).

2.2 To enable discovery and effective use of data in independent sources

While the RDF model and format(s) are a design choice of internal data for applications such as MAD, they are essential for applications that must be able discover and access as soon as possible data from sources not contained in the official DMIS. ADAST is such an application: In response to an alert declared by a responsible authority (e.g., the Central Weather Bureau) warning of an imminent calamitous event (e.g., a severe storm and possible landslides), ADAST proactively notifies people in the affected areas specified by the alert. When the threatened areas include popular national parks, the application must reach not only local residents, but also tourists. The data required for this purpose are likely to be in sources maintained by multiple government agencies and companies. (For example, real-time data on numbers and locations of tourist groups are likely to be in databases maintained by Tourism Bureaus and tour companies.)

The VR serving this and similar applications should provide at least semantics on and links to data critical for the applications to meet their minimal requirements. In the case of ADAST, a minimal requirement is that the contact persons of all tour companies operating in the park(s) and park ranger stations are notified. Hence, a use scenario of the VR is that semantics and links to data on these entities are created and maintained in the storage of the VR during the design and development process of the application. For the same reason, applications designed to support rescue and evacuation operations need data on rescue equipment, hospital and emergency care facilities, and so on in the affected area. Providing the applications and their developers with tools to discover, link and cache such data is a function of the VR serving the applications.

In addition to the basic functions mentioned above, the VR also provides event-notification support. With a few exceptions, sources in and out of official DMIS are passive or interactive databases, sensor webs, web services, etc. Without help, a proactive application such as ADAST must poll their contents and pull from them data on recently posted disaster alerts and specifics about each alert. Having the VR do this work on behalf of all the applications served by it is a way to amortize the associate overhead and thus keep the overhead low. We are developing an Intelligent Active Storage Service (IASS) [14]: This component of the VR will provide applications requesting its service with the capability of monitoring events and conditions defined by the applications in terms of values of data in specified sources and the VR triple store and respond to the occurrences of a event/condition by pushing notifications to designated applications and end-users in ways specified by the applications. By doing so, IASS will turn existing pull-based data sources in part into push-based reactive/active sources.

The version of the VR described in subsequent sections has a simplified IASS, called *client subscribe and notification services*. To avail itself of the services, ADAST first register itself with the services and specifies in its request-for-service the types of disaster alerts (e.g., typhoon, earthquake, debris flows and downpour) about which it wants to be notified and the maximum allowable

delay in notification for each type of alert. These services monitor the authorized information sources where the specified types of alerts and warnings are posted (or published) and notify ADAST when a specified alert is found (or received). Details on the structure, components and operations of ADAST are omitted since they are unimportant to our discussion here and can be found in [9].

3 Structure and design of virtual repository

Figure 1 illustrates the relation between a VR, its *client applications* (i.e., the applications served by the VR) and *client* (*data*) *sources* (i.e., data sources used by one or more client applications): The middle of the figure depicts the structure and components of a VR, which we will describe shortly. The dashed boxes on the top and bottom of the figure enclose its client applications (including MAD and ADAST as examples) and data sources, respectively. The vertical box in the right of the figure shows the generic interfaces such as HTTP GET/POST protocol that enable applications and sources to communicate with tools and components inside the VR.

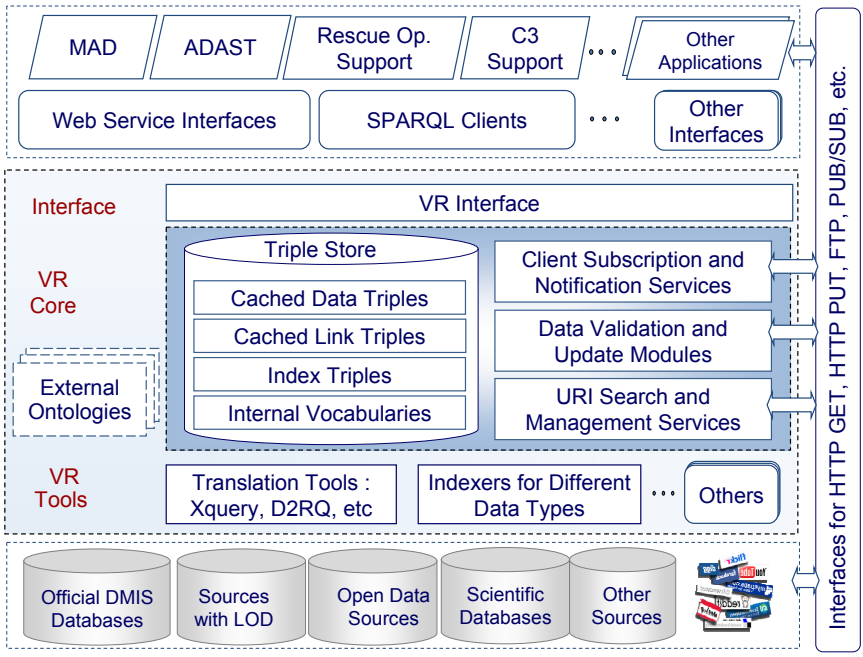

Figure 1: Structure of virtual repository.

In essence, a VR provides its client applications with a development and run-time support environment: By using the tools and processes provided by the VR, a user (i.e., a client application and/or its developer) can create and add as needed semantics and links for parts or all of the data in each source used by the

application and selectively cache and publish the resultant links and LD to a shared common store and thus make them available to other applications.

3.1 Interface and tool layers

As we can see from Figure 1, VR has a layered structure: interface(s) on the top level, VR Core on the middle level and VR Tools on the bottom level. A VR-enabled client application can access data on links, linked data and tools provided by the VR, as well as data in client sources, via a variety of interfaces, including web service interfaces, SPARQL [15] endpoint, etc. The query language used internal to the VR is SPARQL, using which linked data can be queried in ways similar to querying a large database [16]. The VR Interface layer translates all user queries into SPARQL queries and sends the queries to retrieve the required information. In this way, each client application can query the VR with condition specified in SPARQL and set up the returned format or returned information displayed in its own way.

At the backend, the VR Tool layer makes available tools to translate raw data in common exchange formats (including XML, JSON and KML) and data record formats (including XLS and CSV) to linked data in the RDF N-triple format, refereed to simply as triples in Figure 1. Currently, translation tools offered by the VR Tool layer include XQuery [17] and D2RQ [18]. The former can be used to translate XML file to linked data, and the latter can be used to access a relational database as a RDF graph. The tool layer has a recommendation table. Using the table, developers can recommend other existing translation tools to be added the VR tool library or suggest missing tools to be designed and built in the future.

The contents of many files, including media files in formats such as JPEG, TIFF, AVI, MP3, etc. cannot be represented in a RDF format. The Indexers shown in Figure 1 are tools for building indexes for these files based on their metadata and/or user specified contents. The resultant indexes are in RDF N-triple format and stored as shown in the figure.

3.2 VR Core layer

VR Core is the essence of virtual repository: It provides the resources and functions needed to support the VR Interface to client applications and VR Tools for the underlying data sources. The layer is composed of storage components and functional components. The important ones are depicted within the darkened rectangle in the middle of Figure 1.

Before moving on to describe individual components, we note that as the name Virtual Repository implies, VR provides a storage system for information and data. This component is called *Triple Store*. Depending on the client applications, parts or all of the data links or linked data generated by the VR tools are stored and kept up to date in Triple Store. An application can treat Triple Store as a virtual storage, sending all of its requests and dealing with returned data as if all the data are in the component: The fact that some of data are stored only on the original data sources is hidden from such an application.

Client applications can also choose to be unaware of the fact that Triple Store is physically distributed. The middleware places links and LD maintained within the VR core on multiple servers, some of which are physically outside of the areas threatened by the current calamity or in a highly available cloud, so that they remain available and accessible even when the data sources are damaged by the disaster. Some client applications (e.g., MAD) may store their LD pervasively on computers and devices chosen by them. This is a way to reduce the bandwidth and increase the availability of connections required to support end-user access to data. However, these data, unless duplicated in Triple Store, are not accessible by other client applications. Due to space limitation, we defer discussions on achievable availability, responsiveness and other quality of service attributes of alternative distributed configurations of the VR storage to a future paper.

4 Key VR Core components

Again, VR Core has two types of components: storage components and functional components. We now describe the ones shown in Figure 1.

4.1 Storage components

As Figure 1 shows, Triple Store contains at least four types of data: Cached Data Triples, Cached Link Triples, Index Triples and VR Internal Vocabularies. VR provides a space for optionally caching the linked data generated by the translation processes. So, when translating selected data from client sources into RDF triples, a user (i.e., an application or its developer) can choose to store the resultant triples in Triple Store. They are called Cached Data Triples, or simply data triples here. When a user searches data through the VR Interface, VR returns cached triples if they are in Triple Store, instead of assisting the user to find and access the raw data from client sources and translate the raw data to linked data again. Addition to saving the time and overheads of repeated translation, cached triples provide backup and enhance the availability of critical data.

In contrast to data triples, the user may choose to store only the link triples generated during translation. These triples provide the applications with connections to data triples within the VR and in external knowledge bases.

An index triple was generated by an indexer tool to mark a file based on the metadata of the file and/or data specified by the user. A file may have multiple index triples. As an example, let us consider a file containing a video of the terrain and landscape of a highly unstable mountain area shot by an unmanned aircraft before typhoon season as a part of disaster preparedness SOP. The metadata of this video include basic file information such as size and format of the file; geological information (including the location, coordinates, size and type of terrain of the area); weather and operation conditions (e.g., operator, purpose, goal, observed event, analysis report) and so on. These types of metadata are formatted into RDF triples. The resultant linked data on the file enable the file to be found more easily. Depending on the user's choice, the file itself may or may not be cached by the VR.

An important part of Triple Store is VR Internal Vocabularies, or Internal Ontology. Today, numerous ontologies, including DBpedia [19] and SUMO [20], are available. They define and provide representation of concepts and objects (entities) and relationship among them in diverse domains. Figure 1 calls them collectively external ontologies. VR makes use of them as much as possible. The VR Internal Ontology includes only the terms that are required by some client applications but cannot be found or are not defined appropriately in any of the commonly used external ontologies. As an example, when a user uses VR translation tools to translate information on shelters in a city to linked data, a URI is needed to represent the street name part of the address of a shelter. If the user cannot find an existing URI to represent the street, he/she creates an URI of the street and stores the new URI in Triple Store as a VR internal vocabulary with the help from the URI Management Service.

Some or all of the VR internal vocabularies may be valuable to web ontology communities or developers. For this reason, VR also provides a GUI and API functions using which ontology developers can access the vocabularies and evaluate them.

4.2 Functional components

The right part of the rectangle in the middle of Figure 1 encircles some of the functional components provided by the VR Core for both internal and external use. Earlier, we have already mentioned Client Subscription and Notification Services, hereafter, referred to as CSS and CNS, respectively. These services enable client applications and sources to publish and subscribe information to each other. VR provides a web UI and an API function, using which a user can register with CSS as a subscriber, providing the service with specifications of the required data, designated applications and end-users to notify and one or more trigger conditions. When CNS detects a condition in the trigger condition list is met, it retrieves the required data and sends the data as a notification to the subscriber and designated client applications or end-users. With CSS and CNS, client applications do not need to periodically check the data in VR to see whether the data have been updated. Instead, the applications can check the data once they receive notifications from CNS telling them that subscribed data is updated.

Data Validation and Update Modules (simply called DVM and DUM, respectively) are also key components of VR core. For data stored in Triple Store, DVM is responsible for validating the consistency of internal data with external data stored in the original sources. When a user decides to store the RDF data of the client application in Triple Store, the user specifies a valid (time) interval, or simply valid time, for each triple or set of related triples. A triple is considered expired and hence invalid when time elapses beyond its valid interval. As an example of where this is important, we note that a client application such as ADAST must examine the date or valid time interval of each alert. If the alert has already expired, it should not take any action as a response to the alert.

In addition to specifying a valid time interval for stored triples or sets of triples, a user may also provide parameters specifying when to update or not to update when the data is no longer valid. DUM updates the store data in VR based

on the policy defined by such user-specified parameters and in the absence of parameters specifying otherwise, update the data by default.

The bottom box in the VR tool stack depicts URI Search and Management Services, called USS and UMS respectively. The former is provided to assist the users in their search for URIs, while the latter helps to manage (i.e., add, delete, and modify) internal URIs.

During a translation process, when a URI is needed to represent a term, the user (or a translator program) can use USS to search external ontologies for possible choices. USS provides the user with a UI and an API function for this purpose. Once USS receives a searched term via the UI or API function, the tool connects to existing ontologies in turn, queries each of them for URIs based on the search term and integrates URIs from the search into a list as the result of the search. In this way, USS enables the user to treat all external ontologies as a single source and thus avoid the need to search them individually manually.

After receiving a URI list from USS, a user can select a URI from the list to represent the term for which the search was performed. When none of the existing URIs in the list is inappropriate for a term, UMS assists the user to create and manage a new URI. UMS is part of the interface of VR Internal Vocabularies. Every term created by UMS is stored in the VR Internal Vocabularies. UMS allows both manual and automatic addition of new URIs. Specifically, the GUI of UMS lets ontology developers to manually classify and store new URIs offline. UMS also provides API functions using which a client application can request automatic addition of one or more URIs into VR Internal Vocabularies. Both USS and UMS work with VR Translation tools and Indexer.

5 Summary and future work

This paper describes the structure of a virtual repository (VR) designed to present to its client applications and users linked-data views of data contained in one or more sources that may or may not support linked data. Serving as a single data source, a VR enables its clients to be benefited from linked data despite their use of legacy databases and web services. The resources, tools and services made available by the VR aim to ease the tasks of transforming raw data from client data sources into linked data. In this way, the VR serves developers as a component of a development environment for the implementation of linked data enabled applications for disaster preparedness and response.

Specifically, previous sections described the three-level structure of VR. The top level is the VR Interface that enables applications or users to query for required information from the VR using commonly used interfaces. The middle level VR is composed of storage components and functional components. The storage components form a virtual repository of links and cached linked data. The functional components provide services including event notification, data management, URI management, etc. At the bottom level, VR tools assist the translation of raw data from client sources into linked data.

Much work remains to be done. Existing open sources tools and components gave the VR effort a running start. In the process of extending VR tool library and

key components and completing a fully functional VR prototype, we will produce new tools and components and contribute them to the open source resource pool. Examples include the growing collection of URIs representing disaster preparedness and response concepts and objects in VR Internal Vocabularies, as well as functional components that support the VR services presented earlier.

In addition to MAD and ADAST, we plan to develop as case studies other representative disaster preparedness and early response applications that share a VR. They are needed, in addition to the VR prototype, for the purposes of demonstrating convincingly the merits and effectiveness of the VR approach to design and development of linked-data enabled applications and services.

We mentioned earlier that we will enhance CSS and CNS, redesigned them if needed, in order to turn them into an intelligent active storage service and a VR into an active/reactive information system. Until now, we have made ad hoc architectural choices for the distributed Triple Store. Now that the design and implementation of the prototype is well underway, we will begin to flesh out the alternative choices of distributing the contents of the Triple Store on available VR system resources and client data sources and evaluate their relative merits in terms of quality of services parameters such as data availability and responsiveness.

Acknowledgements

This work was supported by the Academia Sinica thematic project OpenISDM. The authors wish to thank Wen-Ray Su, I-Liang Shih, Shang-Yu Wu of Taiwan National Research Center for Disaster Reduction (NCDR) and many Co-PIs of project OpenISDM for their critiques and suggestions on this work.

References

[1] Murphy, R. R., A national initiative in emergency informatics. *Computing Community Consortium*, 2010.
[2] Bizer, C., Heath, T., & Berners-Lee, T., Linked data-the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, *5*(3), 1-22, 2009.
[3] Bauer, F., & Kaltenböck, M., Linked Open Data: The Essentials, 2011
[4] Borges, M. R., de Faria Cordeiro, K., Campos, M. L. M., & Marino, T. Linked Open Data and the Design of Information Infrastructure for Emergency Management Systems.
[5] Silva, T., Wuwongse, V., & Sharma, H. N. Disaster mitigation and preparedness using linked open data. *Journal of Ambient Intelligence and Humanized Computing*, 1-12.
[6] Schulz, A., et al., Integrating process management and LOD to improve decision making in disaster management.
[7] LOD2 Project, http://lod2.eu/WikiArticle/Project.html
[8] SIMILE Project, http://simile.mit.edu/
[9] Lai, Y. A., Ou, Y. Z., Su, J., Tsai, S. H., Yu, C. W., Cheng, D., Virtual disaster management information repository and applications based on

linked open data, IEEE International Conference on *Service-Oriented Computing and Applications (SOCA)*, pp.1-5, 17-19 Dec. 2012.
[10] Berners-Lee, T., Design issues: Linked Data, http://www.w3.org/DesignIssues /LinkedData.html
[11] Open data sites, http://www.data.gov/opendatasites
[12] OpenData.tw, http://www.opendata.tw/gov-data/tpe-od-platform/
[13] Sheridan, J. and J. Tennison, "Linking UK government data," Linked Data on Web Workshop, 2010.
[14] Lee, C. R., Y.Z. Ou, C.W. Yu, S.H. Tsai, F.R. Chern, J.W.S. Liu., IASS: Intelligent Active Storage System, Work-in-Progress Presentation at RITMAN Workshop, co-located with SOCA, Dec. 2012.
[15] SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/
[16] Feigenbaum, L. 2008, *Why SPARQL?,* http://www.thefigtrees.net/lee/blog /2008/01/why_sparql.html
[17] XQuery 1.0: An XML Query Language (Second Edition), http://www.w3.org/TR/xquery/
[18] D2RQ, http://d2rq.org/
[19] DBpedia, http://dbpedia.org/About
[20] Suggested Upper Merged Ontology (SUMO), http://www.ontology portal.org/
[21] OpenISDM Project, http;//openisdm.iis.sinica.edu.tw/