A Design Pattern for Information Sharing in Medical Emergency Response to CBRNE Events

Robert Dourandish
Quimba Software
San Mateo, United States
bob@quimba.com

***Abstract* - Responding to large-scale medical emergencies tends to quickly overwhelm a single agency's recourses and demand multi-jurisdictional response. As the number of responder organizations grows so does the importance and complexity of effective, efficient, and timely information sharing. This is particularly true in the context of operations that may demand collaboration between military and civilian agencies, such as responding to Chemical, Biological, Nuclear, or Explosive (CBRNE) events. With the addition of digital infrastructure and wireless data networks as intrinsic Incident Command tools, a significant barrier to efficient information availability and dissemination is the tiered, multi-domain access paradigm that is typically employed by response agencies, particularly military organizations. While treaty-based response protocols can successfully create digital shared domains, information sharing, particularly in an automated or software-driven fashion, remains exigent. Fundamentally, the challenge belies in the need for heterogeneous system infrastructures and architectures to rapidly fuse in an adhoc fashion, a task that raises wide ranging technical challenges. The cross-domain access mitigation is particularly important in the context of medical information since providing timely care has to be balanced against patient privacy and, in events that may involve biological or nuclear agents, against the best interest of the community at large. Asynchronous Web Services offer a practical solution to the problem of multi-organizational information sharing in the context of time-critical medical emergency response. However, asynchronous operations are not a native component of the W3C standards and while a number of approaches have been suggested, none meet the security and privacy requirements of medical emergency response. This paper describes a design pattern that addresses many challenges of deploying web services in support of information sharing processes across heterogeneous domains in the particular context of the medical component of large-scale multi-jurisdictional emergency response.**

Keywords - CBRNE, Collaborative Emergency Medical Response, Service Oriented Architecture, information Sharing

I. INTRODUCTION

Medical emergency response is inherently a collaborative, multi-organizational operation where patients are cared for in a continuum, by professionals who specialize in various aspects of rescue, treatment, and rehabilitation. In most emergency operations responding organizations belong to the same jurisdiction and, as such, work and train together on a regular basis. This level of exposure helps local responders develop a familial sense about how to best support each other to ensure a successful operation. However, as the incidents grow beyond a single jurisdiction, such as in the case of CBRNE events, the response operation requires inclusion of mutual-aid resources that can only relate to local responders via protocol [1].

Medical component of collaborative response to large scale incidents is an arduous task due to a number of unique properties. For example, while delivering rapid and appropriate care is absolutely vital to victim survival, it must be done in compliance with privacy laws in the response jurisdiction. Furthermore, a complex set of legal and case laws stipulate information dissemination guidelines to other sources, such as the press or law enforcement agencies, that have a legitimate mandate to protect the community. The latter is particularly relevant in incidents involving biological agents with high risk of spreading through human contact, when the need to identify and locate specific individuals may be the key to a timely or effective response. Other unique properties of medical emergency response are the incident tempo that can range from minutes to months; impact on a single individual or the global citizenry; The scope of the incident that can quickly create a chain reaction straining social services; and the secondary societal impact that can inflict severe financial damage to industries such as travel.

Given these attributes, a key success factor in effective large scale medical emergency response is managing information distribution and access amongst the myriad of responders and stakeholder. Due to the increased availability of reliable and persistent digital infrastructure in emergency operations, automated information sharing holds the promise to offer significant benefits in joint operations [2][3][4].

Here we describe a design pattern that eliminates much of the a priori work required in implementing digital data sharing eco-systems. In particular, it will obviate much of the treaty-based agreements in information sharing, such as common servers and data formats. This paper first reviews the background of our research as well as the relevant related work to date. We then focus on the technical implementation details.

II. Related Work

For the past several years, we have been researching massively scaled, multi-jurisdictional, automated information sharing infrastructures, with a focus on emergency response. The discipline is a uniquely challenging example of a time-sensitive, complex, distributed, and networked eco-system because of the extremely broad range of competencies, technologies and resources of network participants. Therefore, as response operations grow in scale, the task of maintaining a shared informational framework becomes increasingly difficult.

While it is possible to create treaty-based, umbrella digital shared domains for a group of response organizations, a Service Oriented Architecture (SOA) is the only logical choice to create a common operational environment that does not require prior technical or policy agreements between all participants [2][5]. Furthermore, the emergency response domain has a number of properties that ideally fit Service Oriented Architecture as the architectural underpinning of a field-deployable information management framework. A number of these properties are:

- Domain is extremely fragmented, with participants having a wide range of capabilities, training, and resources;
- There are key regulations in the US and Europe that limit how health care data is managed and shared. The US Health Information Portability and Accountability Act (HIPAA) regulates data sharing across domains (i.e., police and hospital, or state and federal) as well as temporally along the continuum of care;
- In most cases response must be rapid and decisive, with minutes making the difference between life and death; and finally
- There is frequent need to consult with specialists and other experts on cases such as poisoning or CBRNE response.

The SOA is the only approach that would allow all responders to use their existing infrastructures while cooperating via the wide-area connectivity afforded by the Internet. Using IP-based networks inside the firewall, also a common practice, allowed us to implement the services without the need to distinguish where they were running (e.g., inside or outside an organization’s network) and secure the communication using standard strategies such as creating a Virtual Private Network (VPN) to support each incident.

Employing a Service-Oriented Architecture also solves the challenge of duplicating the infrastructure underpinnings, particularly with respect to adhoc coalitions that are formed in response to specific incidents. Service Oriented Architecture alone, however, does not address the single, most prominent attribute in emergency response, Asynchronicity, which is amplified exponentially as the number of responders increases. Asynchronous operations, however, are not a native component of the W3C standards. This paper describes a Design Pattern that implements a transaction-based, secured, verifiable, and auditable extension to the W3C Web Services Standards.

Exploiting technology in medical emergency response is not a new research topic. The field of practice largely grew out of the desire to reduce response times as detailed in [3][4], with a current survey of the field presented in [6].

In terms of CBRNE response, most jurisdictions use technology to support high-risk response operations. Almost all agencies also carry some variation of sensors to detect presence of CBRN agents on the scene and there is a desire to couple wireless-capable sensors with real-time analysis. Numerous modeling and simulations activities are currently underway to support medical response. Noteworthy in context are EPA’s MENTOR-2E [7], a collection of models that use an integrated, mechanistically consistent source-to-dose-to-response framework to quantify inhalation exposure and doses resulting from emergency events; the Integrated Weapons of Mass Destruction Toolkit (IMWDT) that is used to support our research [8], and the DHS effort to create models of bio-terrorism risk assessment [9]. Active research also includes study of Time Critical Information Services [10]. Furthermore, research is in progress to develop methods of introducing domain expertise in emergency response, for example via Ontology [11][12]. Given the significant collaborative nature of CBRNE response, and large-scale emergency response operations in general, relevant research also includes collaboration and cross-domain information sharing. Of particular relevance are efforts reported in [12], investigating use of Ontologies as knowledge representation instrument in the emergency response domain to enhance automated information sharing. Finally, CBRNE events almost always require multi-jurisdictional response, possibly including support from the military. Research in Command and Control in multi-organizational mission operations includes work reported in [6][11][12].

III. Design Pattern

This pattern is designed to support tactical operations between networked cooperative participants. As visually depicted in Figure 1, the pattern has two major components: Nodes and Transaction Object. A node, synonymous with a “server”, is a collection of web services that are cooperating in support of a mission. The transaction object is used to create and maintain a persistent state throughout the networked environment and to exchange information.

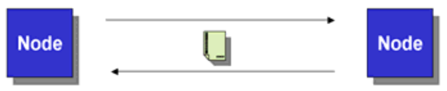

Figure 1: Communications Pattern

The key is that the node must support a default connection service that accepts a transaction as the only argument. The default service is one that is activated when the node is referenced using a normal http or https reference. In networks with high security considerations, the connection service may be "hidden" behind a specific socket [1], and the network may further require human intervention to receive the connection specifications. The node itself may be architected using any number of patterns that utilize an Edge Service. A sample pattern is shown in Figure 2. As depicted, the transaction arrives, **(1)**, and is received by the node's Edge Service, **(2)**. The service will immediately off-load the transaction to a staging area, **(3)**, where it will wait for processing. The rapid off-load enables significantly higher performance by the Edge Service, particularly if connection is made over UDP [2]. In our implementation the staging area is a directory on disk. While the staging area could be implemented in a database, we found the overhead of database write operations an unnecessary burden in this specific case.

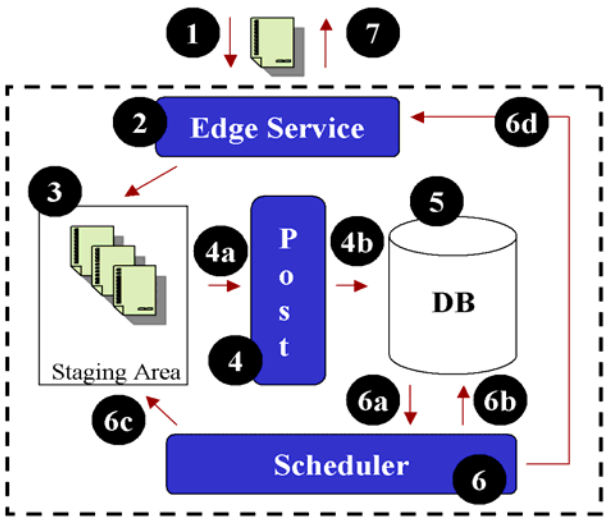

Figure 2: Components

In our implementation redundancy, not shown in figure, was provided via RAID-0 (mirroring) of disks and audit trail, also not shown, can be implemented using the disk logs. A post process, **(4)**, will loop over the staging area. New transactions are pulled off, **(4a)**, from the staging area written, **(4b)**, to an operational database, **(5)**. The post process may perform additional security or assurance operations before inserting the transaction in the database. Note that, as discussed in the Section entitled 'Transaction Processing', the transaction may include encrypted components. A third process, a scheduler, **(6)**, loops over the operational database. This process will examine, **(6a)**, each transaction and dispatches it to an appropriate internal service for processing. Based on results, the system may update the transaction in the database, **(6b)**, may place a new transaction in the staging area for re-posting, **(6c)**, or may send an outbound transaction, **(6d)**, to the edge service, **(2)**. This outbound transaction is, most likely, either an acknowledgement or response to a transaction and is sent to the originating node, **(7)**. Since this process is duplicated on each node, a collective system state is established and maintained.

A. *Database Tables*

Two database tables are used in support of transaction management; a *Queue* table, and a *Transaction History* table. Both have the same schema, shown in Table 1, that matches the transaction object.

TABLE 1: TRANSACTION TABLE

Column	Meaning
TID	Transaction ID (originator)
TRefID	Transaction ID (recipient)
Tchannel Priority	Self-explanatory
Ttype	See Transactions Processing
Tcode	See Transaction Processing
Tformat	TXT, XML, Binary, Etc.
Tlength	Length of this Transaction
TinitDate	Self-explanatory
TlastTouched Date	Self-explanatory
TcaseInfo	Additional Reference Number or designator, if needed.
TCRC	Self-explanatory
Treserved	Self-explanatory
Tdata	Data Segment. Variable length See Transactions Processing
Tsender	Self-explanatory
Ttarget	Self-explanatory
TStatus	Current Status. See Transactions Processing.

To ease implementation and improve performance the Queue Table can further be sub-divided into three tables, *Low Priority Queue (LPQ), Medium Priority Queue (MPQ)* and *High Priority Queue (HPQ).*

[1] Security operations are generally easier when using a single Edge Service vs. a directory approach that publishes numerous services supported by the node. This, however, does not reduce the security burden beyond easing connection security issues.

[2] For completeness: While UDP connections are not lossless operations, they are the ideal connection type for peer-to-peer operations since the server will not need to maintain a persistent connection. This is property is particularly useful in massively scaled, peer-to-peer networks, such as emergency response where the nature of an incident will determine the response network and a-priori network design is fundamentally futile.

B. *Transaction Processing*

As compared with traditional client-server, transaction processing in a distributed environment and dynamic peer-to-peer connection is somewhat more complex.

As with a client-server environment, a *Transaction ID*, **(TID)**, is assigned to each transaction at initiation time by the node that is acting as the "sender". The TID alone, however, is not enough of a tracking instrument in a distributed system since it is not possible to guarantee system-wide uniqueness. To address the issue the pattern employs a second field, a *Transaction Reference ID*, ***(TrefID)***, that is assigned by the target of the transaction, or the node that is acting as a "recipient". The combination is guaranteed to uniquely identify the transaction system-wide. Actions are requested through the combination of *Transaction Type* **(Ttype)**, and *Transaction Code* **(Tcode)** determines the action being requested, by the sender or an acknowledgement or reply to the request, by the recipient. Transaction codes and types are shared by all nodes and are part of the foundational core system that is shown Figure 2, and a sample code is shown in below.

```
// GENERAL STATUS CODES (00)
int GENINIT=0x0000; //init
int GENFINOK=0x0001; //finished w/o err
int GENFINERR=0x0002; //finished w/err
int GENUNRESP=0x0003; //unreponsive proc
int GENORHPAN=0x0004; //orphaned process
int GENWAIT=0x0005; //Waiting.
int GENREADY=0x0006; //txn ready to run
int GENPRIHOLD=0x0095; //priority hold
int GENADMINHOLD =0x0096; //adm hold
int GENSECHOLD=0x0097; // security hold

//DAT STATUS CODES (02)
int DATBegin=0x0200; //query begin
int DATEnd=0x0299; //End of Data
int DATWait=0x0201; //Waiting for data
int DATACK=0x0202; //Data-related ACK
int DATQuery=0x0203; //new query
int DATSend=0x0204; //Ready for data
int DATReceive=0x0205; //Rdy to send
```

All necessary parameters are transmitted through the *Transaction Data* **(Tdata)** component of the transaction object. This component can be encrypted. The ability to encrypt data and parameters, coupled with the singular edge service, substantially addresses many of thorny cross-domain data access and information sharing issues – this is simply accomplished by the intermediary node, placed between higher and lower echelon nodes, duplicating segments of an incoming transaction, then forming a new transaction and submitting that transaction for processing to a lower echelon node for processing. The two transactions can be related by utilizing either the *Transaction Case Info* **(TcaseInfo)** field, or as another encrypted element in the data component of the transaction (Tdata).

C. *Transaction History and Audit Trail*

A Database Trigger, shown below, accurately manages the transaction history table.

```
create trigger txnq after insert on txnhistory
for each row begin
delete from lpq where tid = new.tid; -- delete it if exists
insert into lpq values (new.TID,…);
end;
```

A similar trigger is implemented to execute after update. The net effect of the triggers is that a copy of the *old* transaction (before the queue is updated) is inserted into the *Transaction History* Table and the entery in the *Queue Table*, e.g., *LPQ*, is overwritten with the most current status. Querying the transaction history table on a single node (based on the TID) will provide an audit trail on that node. To produce a system-wide audit trail all nodes must be queried.

D. *Scheduling*

As shown in Figure 2, a scheduler loops over the transaction queue(s) – tables – in the production database. If the *transaction status* **(TStatus)** is not any of the hold status codes, i.e., priority, administrative, or security holds, the transaction will be dispatched, as discussed in the next Section. The scheduler will update the status code before dispatch. This act creates an entry in the transaction history table, providing an audit trail. The Scheduler has a discrete component that will dispatch – execute – code that perform tasks necessary for, or requested by, other nodes via the transaction object.

E. *Dispatch*

Tasks, i.e., responding to a request for data, are accomplished by dispatching the transaction to a handler code. Handler codes do not have to be determined in advance and a default handler is available as part of the core foundation. Each handler may be a Web Service inside the fire wall (behind the Edge Service) and is not visible to outside callers. Handlers can also be implemented in different forms, such as libraries or class that are loaded or linked dynamically at run time. This allows each node full capability to customize handling of each transaction.

Handlers are specified via a database table in the production database. This table can be stored in a separate or more secure database, or be encrypted for additional security. Table 2 shows the schema for the handler table in the production database.

TABLE 2: HANDLER TABLE SCHEMA

Column	**Meaning**
CMDGroup	Transaction Type
Handler	Handler Name
CallType	Handler Type

Each handler is tied to a transaction type via the *Command Group*, **(CMDGroup)**, field. The *Handler Name*, **(Handler)**, specified the programming name for the handler. The final database column, **(CallType)**, specifies the specific handler type, for example a servlet, a class, or a dynamic library. An actual sample of the table is shown in Table 3. Note the relationship to the sample code above.

TABLE 3: HANDLER TABLE

CMDGroup	Handler	CallType
CNC	C2	1
DAT	dataManager	1
GENACK	GENACKClass	0
ICH	InfoChain	1
INIT	INITClass	0
RDR	fileReader	1

The dispatcher follows a relatively simple algorithm: it determines, based on the transaction type, which handler to invoke. It then applies the appropriate call type sequence based on the information in the handler table (CallType), and passes the entire transaction object to the handler code. Nothing else is passed on, expected, or accepted as a parameter. This approach drastically simplifies not only management and maintenance of the database support for scheduling and dispatch operations, but also measurably reduces the complexity of extending the environment through custom code.

IV. CONCLUSION AND FUTURE WORK

Asynchronous operations are the cornerstone of all but the simplest of collaborative tasks. As such, they are an important component of joint operations, particularly in specialized, non-persistent or mission-based tasks such as military-civilian disaster response. Unfortunately, asynchronous operations are not part of the W3C standards specifications for web services. In this paper we discussed a transaction-based design pattern to implement massively scalable, cooperating asynchronous web services. The pattern was developed and tested as part of a research project focused in the military-civilian information sharing and has, thus far, proven functional, stable, and consistent. Consistency is a large component of a useful pattern since building extensive exceptions will eventually render the pattern useless. We have not thus far encountered a situation where the pattern needed to be augmented or mediated through exception processing. Our future work in this regard is now shifting to application of the pattern to additional domains such as simulation and training.

ACKNOWLEDGMENT

The various efforts culminating in this paper were supported by a number of small business research grants awarded by various United Sates federal agencies over a five year period. Specific details of the awards, and their contribution to this research, may be obtained by contacting the author.

The views presented in this paper are those of the author and do not necessarily reflect the opinion of the funding agencies or the United States government.

REFERENCES

[1] J. Barbera, and A. Macintyre, "Medical Surge Capacity and Capability: A Management Sytem for Integrating Medical and Health Resources During Large Scale Emgergencies", US Department of Health and Human Services, 2nd Edition, pp. ix-xii, September 2007.

[2] R. Dourandish, N. Zumel, and M. Manno, "A Design Pattern for Asynchronous Web Services in Secure, Cross-Domain, Information Sharing," Proc. MILCOM 2007, Orlando, FL, 2007.

[3] B. Schooley, T. Horan, and M. Marich, "User Perspectives on the Minnesota Inter-organizational Mayday Information System," in AMIS Monograph Series: Volume on Information Systems for Emergency Management, Van De Valle and Turoff, Eds.: IDEA Press, 2008.

[4] J. Peters, and B. Hall, "Assessment of ambulance response performance using a geographic information system," Social Science and Medicine, vol. 49, 1999.

[5] H. Adams, "Asynchronous operations and Web services, Part 1 and 2," IBM Systems Journal, available on ibm.com/developer, Last accessed December 2010.

[6] B. Schooley, T. Horan, and M. Marich, "Integrated Patient Health Information Systems to Improve Traffic Crash Emergency Response and Treatment," Proc. 42nd Hawaii International Conference on System Sciences, 2009.

[7] Cuncil for Regularory Environmental Modeling, Model Report, on cfpub.epa.gov/crem/knowledge_base/crem_report.cfm?deid=193583, last accessed December 2010.

[8] R. Dourandish, N. Zumel, and M. Manno, "Automated Military-Civilian Information Sharing," Proc. 2nd IEEE conference on Situation Management, MILCOM 2006, Washington, D.C., 2006.

[9] The National Research Council," Department of Homeland Security Bioterrorism Risk Assessment: A Call for Change," National Academies Press, ISBN 0309120292, 2008.

[10] T. Horan, and B. Schooley, "Time-Critical Information Services", Communications of the ACM, 50, 3, 73-78, 2007.

[11] M. Turoff, C. White, L. Plotnick, and S. Hiltz, "Dynamic Emergency Response for Large Scale Decision Making in Extreme Events," Proc. of the 5th International ISCRAM Conference, Washington, DC, 2008.

[12] R. Dourandish, N. Zumel, and M. Manno, "A Design Pattern for Automatic Generation of Web Services from Domain Ontologies," Proc. 3rd International Conference on Web Information Systems Technology (WEBIST), Barcelona, 2005.