International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:5, 2008

Service-Oriented Architecture for Object-Centric Information Fusion

Jeffrey A. Dunne, and Kevin Ligozio

***Abstract*—**In many applications there is a broad variety of information relevant to a focal "object" of interest, and the fusion of such heterogeneous data types is desirable for classification and categorization. While these various data types can sometimes be treated as orthogonal (such as the hull number, superstructure color, and speed of an oil tanker), there are instances where the inference and the correlation between quantities can provide improved fusion capabilities (such as the height, weight, and gender of a person). A service-oriented architecture has been designed and prototyped to support the fusion of information for such "object-centric" situations. It is modular, scalable, and flexible, and designed to support new data sources, fusion algorithms, and computational resources without affecting existing services. The architecture is designed to simplify the incorporation of legacy systems, support exact and probabilistic entity disambiguation, recognize and utilize multiple types of uncertainties, and minimize network bandwidth requirements.

***Keywords*—**Data fusion, distributed computing, service-oriented architecture, SOA.

I. Introduction

It can be argued that the value of a service-oriented architecture (SOA) is best illustrated by its applicability to a wide range of business applications. Such breadth is easily seen by simple Internet searches for exemplar SOAs, and outlined by the overviews and case studies found in standard reference books such as those by Erl [1], [2]. While many people often associate SOAs with web services, SOA encompasses a broader set of capabilities. We will not provide an overview of the requirements or capabilities of SOAs in this paper, as they are readily available elsewhere (a simple overview, for example can be found online [3]). Rather, this paper describes an SOA approach for facilitating object-centric data fusion, in which a single "thing" – such as a particular cargo ship, a computer system, an individual, or even a category of persons – is the focal nexus of the information of interest.

Data fusion applications can be described as falling into one of two categories. Homogeneous data fusion applications take similar types of information and combine them in a way that best associates and reconciles the information. Basic target tracking[1] is an example. Multiple position reports of similar or identical nature are evaluated, and establish, or are assigned to, tracks based upon fusion algorithms. Heterogeneous data fusion, on the other hand, stitches together multiple types of information in order to formulate a coherent picture. Tools for establishing situational awareness are examples of these. Position/track reports, object characteristics, behavior prediction, etc. are all brought together to produce a fusion product. A wealth of examples can be found in fields such as law enforcement, maritime domain awareness/situational awareness, etc.

Naturally, most real systems do not fall cleanly into one category or the other. For example, consider the case of syndromic surveillance in which one is searching for early indicators of epidemics. The combination of similar data from multiple hospitals to establish average admittance rates, or perhaps multiple pharmacies to determine net over-the-counter drug sales, represents homogeneous fusion, whereas the combination of the two sources is a heterogeneous data fusion task. Examples of such fusion activities and associated requirements/challenges can also be found in the literature [6]-[8].

To best illustrate the utility of this architecture, consider an application in which various sensor measurements and historical database archives provide estimates of one or more attributes of an object. As an example, consider the desire to specifically identify and classify a military aircraft that is expected to soon launch, where the following information sources are available: a) a low-resolution, black-and-white reconnaissance photograph in which the aircraft is present; b) an eye-witness report of a certain limited set of aircraft markings; and c) databases identifying local airfield capabilities (perhaps fueling equipment, runway lengths, etc.). While the black-and-white photograph might provide reasonable estimates on fuselage length or wingspan, these are insufficient to identify a particular aircraft. Similarly, the limited aircraft markings might aid in identifying the nationality of the asset, but themselves do not indicate the exact type of plane. Lastly, knowing the type of fuel the

Manuscript received April 30, 2008.

J. A. Dunne (phone: 443-778-4722; fax: 443-778-5950; e-mail: jeffrey.dunne@jhuapl.edu) and K. Ligozio are with the Johns Hopkins University Applied Physics Laboratory, Laurel, MD 20723 USA.

[1] The term "basic" here specifies target tracking that focuses solely upon the process of best estimating tracks from noisy data, as opposed to systems that seek to combine such tracks with other types of information. "Basic" does not at all suggest simplicity, however. The development of effective trackers is, even after decades of work, still an ongoing research effort. The interested reader is referred to the abundance of research papers, as well as standard references such as those by Bar-Shalom [4], [5].

aircraft uses, or being able to establish limitations on the required runway length, themselves do not permit unique identification. However, whereas each individual datum is itself insufficient, the fusion of these data likely would enable more exacting classification.

This type of data fusion is of interest in many domains, of which military capabilities as described above are just one example. Another, which the prototype below is more closely designed towards, is the fusion of human identifiers (in this case name and social security number) to support automated processes for the disambiguation of individual databases.

The description in this paper covers the exchange and management system to support this type of object-centric information fusion, i.e. the supporting architecture. It does not address any details related to specific applications, such as specifics of particular data archives, any implemented fusion/correlation algorithms, or other case-specific business process chain details. The paper is divided into several sections. Section II summarizes the design requirements and expectations that motivated this information architecture. Section III provides an overview of the architecture in terms of its basic elements and how they interact. Section IV describes a prototyped Microsoft .NET implementation with some basic examples of how the architecture was realized in terms of specific interfaces. Finally, a summary is provided in Section V.

II. Design Requirements

As mentioned in Section I, the SOA presented in this paper was designed in support of a project in which disparate pieces of raw data are assessed in order to reduce the ambiguity associated with a particular object under consideration. Such data might come from a database or a measurement, but in either case are likely to contain multiple sub-pieces of data related to individual object attributes. In the previous examples, the photograph of an airplane (the measurement) gave estimates for two object attributes, wingspan and fuselage length. Similarly, a record from a database of employee information might contain object attributes such as name, age, duration of employment, salary, etc. Within this architecture, each attribute is treated as the atomic unit for consideration. As described later in this paper, combinations of such atoms (molecules, if you will) can also be stored and manipulated, but here we will focus on atomic attributes for pedagogical reasons.

These estimates of attributes can be used in a variety of ways. As a starting point, one might simply compare values with historical data in one or more archives to identify what known objects have matching characteristics. A more in-depth analysis might rank, with some appropriate weighting, a set of known objects based upon how many characteristics matched, and how closely. Still more sophisticated systems might use knowledge about the sensors and environmental conditions in such matches. For example, suppose that a high resolution photograph in good illumination showed a yellow sinusoidal pattern marking on the side of an aircraft, and a low resolution image in poor lighting displays a brown streak. Without knowledge of the sensor capabilities and conditions of the measurements, these features might be considered incompatible, whereas a more informed algorithm could recognize the possibility that they represent the same marking.

Very complex systems might seek to use inference to improve performance. Suppose the presence of a propeller on a plane in the photograph can be used to place limits on the maximum speed of the aircraft. Representation of such information within the system adds value by permitting the exclusion of certain objects from consideration (in this case, excluding all airplanes known to be able to travel faster than the speed of sound).

The overall goal of this architecture is to support a broad range of analyses based on the idea that there is a single object to which observable attributes apply. Such objects might be concrete (e.g. a specific physical object) or abstract (e.g. a conceptual object or meaningful categorization of physical objects), but in either case attribute estimates (direct or inferred) must be addressed with the flexibility described above. These analyses could aim to produce a variety of outputs, as illustrated by the following questions (from most to least direct):

1. What/who is this? (comparison against a potentially large set of enrolled objects to identify the best match)
2. Is this ____? (comparison against a single object to assess similarity)
3. Has this object been observed before/How likely is it that this object has not been observed before? (comparison against prior measurement sets)
4. Is this object atypical? (assessment of characteristics against a norm standard)
5. What other characteristics should be measured to improve classification? (evaluation of knowledge dimensionality)

In addition to supporting the raw data and inference capabilities described above, as well as the ability to answer these kinds of queries, several other requirements were identified. These are briefly summarized in the following subsections.

A. Complex Data

Most legacy database archives contain relatively simple data – strings, numbers, dates, etc. Newer systems are more likely to include uncertainties or other qualifying metadata, but they still represent comparatively straightforward information compared with the actual measurements from which the data were distilled. Even many transducers collect raw data that can be represented in relatively straightforward formats (binary or analog data that can be discretized). However, the input to data fusion algorithms often require more detail, and can come from sensor systems that, rather than distilling measurements down to simple numbers, process raw data into more complex structures. Consequently, this information architecture must be designed to handle data that has a variety of complexities. This is especially important for probabilistic fusion engines. In order to support whatever fusion algorithms are appropriate, the architecture was designed to address multiple complexities in the data:

- Fuzziness: Beyond discrete numbers, strings and Booleans, the architecture must be able to handle approximations, inequalities, etc.
- Structure and higher dimensionality: The system must be able to handle intervals and matrices in order to support compound sets of values such as discontinuous ranges, spectra, multiple string values, etc.
- Uncertainty: The system must recognize and facilitate the handling of uncertainty associated with the data. While simple uncertainties might be addressed under the previous bullet (i.e. using intervals), this is not uniformly the case for all types of uncertainties. Two important aspects that require specific handling are:
  - Likelihood – Likelihood refers to the chance that a measurement of an attribute of an object will match a prior measurement. In some cases the likelihood is high. For example, each time one measures the length of an aircraft, it is likely that it can be meaningfully compared with prior measurements. The likelihood is not as great for an attribute such as the weight of the plane (which would change slightly depending on what was in it), and substantially less so for more variable quantities like the speed of the aircraft, which could change drastically depending upon the conditions of the observation.
  - Applicability – Applicability refers to the degree to which an attribute applies to an object. To continue the aircraft example with a somewhat contrived example, suppose one side of a plane was painted blue and the other side red. The assessment of whether the skin color was blue or red depends on circumstance, so one can assess the applicability of the characteristic as being less than unity (for example, "the skin color is red" is only 50% applicable).

To further elucidate upon the concept of applicability, a more meaningful example is the following. Suppose a particular (previously unseen) person is known to have descended from two ethic backgrounds, A and B. Further, suppose it has been determined that people of ethnicity A have some likelihood of having blue eyes. Now assume that a measurement (perhaps a photograph) shows the existence of a person having blue eyes. If we wished to use eye color as a factor in determining the probability that this was the person of interest, it is necessary to understand not only the likelihood of a person of ethnicity A to have blue eyes, but the degree of applicability of the assignment of that ethnicity to the person. Even if the likelihood of blue eyes for ethnicity A is 100%, the fact that that ethnicity is only somewhat applicable to the specific person plays an important role in making an accurate assessment of the situation.

Lastly, the system must be able to handle quantities not only in terms of specified values, but also as functional relationships to other values. For example, suppose a naval vessel of a particular class can be identified by its length, but that different ships of that class each carried different armaments depending on when it was built. A functional relationship provides the necessary "link" to connect otherwise disparate pieces of information (e.g. knowing the ship class and that it has already fired more than N surface-to-air missiles might now be utilized to uniquely classify the vessel).

B. Legacy Data

For most applications a great deal of data already exists in legacy systems. Consequently, the architecture should be able to interact with and utilize those systems' resources and capabilities. Those capabilities can vary significantly, from systems with substantial computational resources that can support high query and data transfer rates, to older systems that could take minutes or longer to respond to a single request. This might even include non-automated legacy systems, such as processes requiring human action/ intervention. Perhaps more so than any other, it was this requirement for asynchronous operation that makes an SOA ideal for the application.

The requirement to interact with these types of systems is further constrained in that it is assumed that the legacy systems will not necessarily change as a consequence of the data consumer's requirements. Therefore, an intermediate capability must exist that "understands" the nature of the information contained within the legacy system, and can mediate the SOA's requirements with the capabilities of the data source. Such intermediate systems are also essential to facilitate the disambiguation problem arising from the use of multiple independent data sources.

C. Downsizing and Extensibility

As with SOAs in general, it is desirable that the system as a whole be stable against problems with individual services. For example, in a system that seeks to fuse person-related data, one should not be restricted from using height and weight data simply because the service that addresses blood type is down for repair. In other words, since there is no guarantee that all resources will be available all of the time – whether it is due to service-specific issues, network accessibility, bandwidth restrictions, policy/governance issues, etc. – the system must continue to operate when individual capabilities are removed. Similarly, different use cases will dictate how many computation resources can be applied to a situation. The addition or removal of such resources should only impact the efficiency of the system and the scope of the processing that it can perform, not whether the architecture will function.

Of at least equal, but likely greater, importance is that the architecture must be scalable, easily updated, and extensible. This need is generally ubiquitous across all data fusion applications, and is another reason why an SOA approach is desirable, because it provides the ability to seamlessly a) add new data sources as they become available, b) incorporate new fusion algorithms and update existing ones, and c) increase available computation resources. Moreover, such changes must not require subsequent modification to other portions of the system in order to utilize the new resources, and the addition of improvements should not necessitate shutting the system down.

III. Architecture

The following subsections describe the architecture in terms of its essential elements (Section III.A) and their interactions (Section III.B).

A. Architecture Elements

At the core of the architecture is a central database that serves to establish a unique record for each object. For illustrative purposes, throughout this section let us suppose that the system is being used to fuse data regarding military assets. In this case, any particular asset has one or more rows in this Object Index (OI). Additionally, the idea of an object, such as "tanks," can also have a record (making this an index of both concrete and abstract objects), so that one can assign characteristics to the idea of a tank rather than to an individual, specific tank.

If all information was conclusive, an object would have only a single row in the OI. In the absence of such conclusive proof, however, it is necessary to retain the possibility that there are multiple distinct objects. At the same time, though, one still wants to capture the possibility that two objects might be one and the same. The service that fills this need is the Object Equivalency Index (OEI). This service tracks the degree of belief that two records in the OI represent the same thing.

The value of the OEI is illustrated in the following example. Consider a first observation of a tank reporting that it has a 75 mm main gun. In a later observation of a tank – which we think might be the same one, but we are not completely sure – it is seen that the tank has twelve wheels within its tread. A third observation now reports seeing a World War II Comet Cruiser tank (which has 12 wheels and a 75 mm gun), and we want to know if this "third" tank has been seen before. Without recording the possibility that the first two observations were potentially both of the same tank, we would only be able to return two matches with comparatively low measures of confidence (each only matched one piece of information). However, through the OEI, we can recognize and utilize the possibility that those two records actually represent the same object, enabling a fusion algorithm to potentially report a single, higher confidence conclusion.

Although this was not implemented in our prototype, this same OEI can also retain hierarchical information. For example, if one record in the OI represents the category "Comet Cruiser Tank," and another the observation of a particular tank that has some probability of belonging to that category, the OEI could capture this information in order to provide the functionality of inherited properties.

Two other indices within the architecture are the Sensor Index (SI) and the Environmental Index (EI). The former maintains information about the sensors used to collect data. This is important because the details of measurement collection can be crucial in evaluating the data. As an example, consider that different microphones can have substantially different frequency response curves. Knowing what microphone was used to collect a particular audio recording can be essential for knowing how to meaningfully compare that measurement against measurements made with other microphones (or if such a comparison can meaningfully be made at all!).

Similarly, an EI is required for maintaining information regarding the environmental conditions associated with particular measurements. In certain applications it might be necessary to have multiple EIs, or even multiple SIs, depending on the nature of the system's focus. In other cases, perhaps where there are no measurements per se (such as in the disambiguation of data from multiple databases), it is possible that neither index would be required.

The basis of system operation is that there exists an Attribute Service (AS) for each attribute relevant to the objects. For systems addressing fusion for people, one might have a Name AS, a Date of Birth AS, a Social Security Number AS, a Hair Color AS, and so forth. Each of these attributes represent the information atoms described earlier. In cases where attributes are appropriately linked together, an AS might exist for that molecule of information (e.g. a Height/Weight AS), but this will be discussed later.

In general, each attribute has at least one corresponding AS. Similarly, each interfaced legacy data system will have at least one association AS. It is possible to have multiple attribute services for a particular attribute (for example, if there are many services offering to manipulate that kind of information) or legacy system (for example if that system maintains information on multiple attributes).

The primary purpose of an AS is to "know about" a particular attribute – what style of information is required to specify it; how is that information best retrieved, stored, and displayed; how are multiple values meaningfully compared with each other. As such, the primary activity of an AS is homogeneous data matching. Whenever information about a particular attribute is being processed or requested, those actions are carried out by the AS. It is possible to denormalize a system in cases where certain requirements or types of processing would benefit, but such is beyond the scope of this paper. The AS is also the source/repository for Graphical User Interfaces (GUIs) and other interface mechanisms that are used to handle attribute data. A user's thin client actually obtains its GUI's for inputting and displaying specific attribute data directly from the AS.

For attributes that are not tied to an external legacy system, the AS is the repository for attribute data, and therefore will generally have its own databases (that are linked to system indices such as the OI, SI, EI, etc.). When external data is used, the AS serves as the intermediary between the legacy system and the information architecture. It encapsulates all of the logic and processes necessary to access and utilize the external data, to perform any caching or augmenting necessary for system operation, and to interpret external data in the context of the overall system. In this capacity, it retains the information necessary to tie legacy system foreign keys to the OI (and other indices as appropriate). As with most services in an SOA, an AS represents a combination of database technology and software applications.

As alluded to above, an AS does not necessarily always address a single attribute. In cases where sets of attributes are best analyzed jointly, an AS can be developed to perform this function. Consider, for example, the case where one is considering both the height and weight of a person who happens to be both very short and very heavy. One could create a height AS and a weight AS, and perform matching comparisons independently, combining the results within a correlation service (see below). This, however, might not be the most efficient mechanism for system performance, and it would be preferable to treat the set of attributes as the quantum of data and build a dedicated AS for that information molecule. In general, molecular attribute services make sense when a) a group of attributes are always measured, stored, and processed as a logical entity, or b) the exploitation of complex correlations amongst the attributes is more common than working with the individual attributes separately[2].

Perhaps most importantly, attribute services are (in the spirit of the archetypal SOA) instantiated independently of each other, and can be designed to access and utilize other known services. Multiple services for a single type of data can be built and incorporated even when they "compete" with existing services. This enables a competitive "survival of the fittest" approach to system growth where services are utilized (and discarded) based on their demonstrated capability, as evaluated by individual users or via collected/tracked usage and performance statistics.

The attribute services play an important role in the data fusion process, but to use them exclusively for this purpose would result in an exponential number of services (one for every possible combination of attributes to be considered). In many cases, heterogeneous information can be treated orthogonally by assuming that the individual data are uncorrelated, or at least that the relationships between the data can be treated as a separable problem, i.e. the comparison of a height to a height is not directly dependent on a weight (even though the value/meaning of the result of that comparison is affected by the weight). For these reasons, the architecture also includes the concept of a Correlation Service (CS).

A CS takes the results from two or more other services (each of which could be an AS or a CS), and fuses those results using some algorithm. It is, in fact, the correlation (i.e. fusion) algorithm that is the defining feature of any particular CS. As an example, a simple CS might take two results sets[3], one from a comparison of wingspans and the other from a comparison of engine types, and determine what assets best match both measurements. Because this kind of generic fusion approach can be applied as meaningfully to aircraft based on wingspan and engine type as it can for cargo vessels using hull color and the cruising speed, a single CS can replace N^2 combined attribute services.

CSs can also be designed to use more specific knowledge of the data. For example, one could design a CS for the earlier example of comparing height and weight[4] that uses the knowledge that tall people are more likely to weigh more, and so would recognize the increased value of having matched both height and weight for someone who is, for example, very short and heavy or very tall and light.

As new fusion approaches are developed, they can be instantiated into new CSs, and registered within the architecture. Like the attribute services, CSs can "compete," with the most useful and effective ones taking over processes that were originally handled by the less robust services.

The last two elements of the architecture are the Registry Service (RS) and a thin client interface that connects to the various other elements. The RS tracks the presence and locations of available attribute and correlation services, as well as the various indices (OI, OEI, SI, EI), connecting the various services according to their requirements. The thin client interface is what enables a user to take action within the system, and is discussed in greater detail in the next section.

B. Element Interactions

Although the OI and other indices represent the true core of the architecture, the RS and user client are the central elements in coordinating the exchanges of information between the indices and the attribute and correlation services. In order to provide an overview of the interactions of the various elements, we will outline the array of processes involved from startup through the use of the system to identify an object based upon a set of measurements. In this example there will be two attribute services and a single correlation service. For illustration purposes, we will assume that measurements are inputted manually by a user rather than via an automated process.

The first system to initialize is the RS. After that, the OI, OEI, EIs, and SI all initialize, each contacting the RS to alert it that they are active and to provide their respective addresses. Next, each attribute and correlation service initializes and exchanges messages with the RS to provide its address. All of these services may continue to exchange messages with the RS on a periodic basis to confirm that they are still active. Additionally, the services request and obtain the addresses for the different system indices.

Finally the user interface system (UIS) initializes and connects to the RS in order to learn what attribute and correlation services are currently available. In this example

[2] There are, of course, advantages and disadvantages to each approach, and the architecture supports both in order to provide the greatest flexibility and capability.

[3] In most cases (but not all) a results set is a ranked list of comparisons for known objects against the specified parameter(s). In this sentence's example, it might be a list for all military assets asserting the probability that the measurement is in agreement with prior knowledge for that asset based on how closely the measured wingspan agrees with the historically recorded wingspan.

[4] Because the specific comparison mechanism for each attribute is independent, even if the net correlation is not.

the user wishes to enter measurements A and B, and so selects the corresponding AS for each. The RS then provides the UIS with the connection information for those two services.

For the thoroughness of the example, we will assume that this is the first time the UIS is interacting with the AS for measurement A. The UIS connects to the AS, and requests a GUI and data object template for displaying and holding data for measurement A. These are returned to the UIS as a function library for instantiating the GUI and a class for the data. If the UIS has not interacted with the other AS before, it would do the same for measurement B.

The UIS now displays the received GUIs and the user inputs the measurement data for attributes A and B. The UIS now sends that data, in the form of serialized objects, to each AS respectively. For the sake of this example, assume that the user is seeking to identify the object via comparison with previously archived measurements.

Each AS takes the data it receives and begins comparing it against its archives of previous attribute measurements. As necessary, it accesses the sensor and environmental indices to request and receive any information needed to make effective/correct measurement comparisons. Once an AS has completed its internal comparisons, it might (depending on system settings) also contact the OEI, providing a list of object identifiers (as are stored in the OI) and requesting all additional identifiers that might be alternate representations.

During this process, the AS can (again depending on system settings) send periodic updates to the UIS to inform the user of its progress. It might also accept requests from the UIS to query regarding its status, or perhaps to cancel or abbreviate its searching processes, depending upon the sophistication of the capabilities provided by the AS.

Once the AS has completed its processing, it sends a message to the UIS (with a unique results set identifier) indicating that it has completed its comparisons and is awaiting further instructions. Note that it does not automatically send its results to the UIS, as the user may not wish to view them directly, and it is possible that the transfer of the results set could be (depending on the circumstances) bandwidth intensive.

When the alert arrives the user is prompted that one or more sets of results are completed, and s/he can decide what to do with the output. If numerous measurements were being processed, and some are expected to take a long time to process, the user might choose to send a subset of the measurement results to one or more CSs, or perhaps view one or more results sets directly. In this example, the user will wait until both sets are available.

After both sets of measurement comparisons are complete and the user has chosen a CS to use, the UIS sends a set of messages. To the CS it indicates that it is requesting action, and provides a list of the results sets, by identifier, that it is to receive. To each AS it sends the address of the chosen CS and a request to forward the results set. Each AS responds back to the UIS indicating that the request is received, and then another updating message once the data transfer is completed. The CS responds with similar messages to the UIS, one indicating that the request was received, and one for each results set transferred.

The AS does not automatically delete its results set at this time. Once the user confirms via a message to the AS that those results are no longer needed (e.g. after having decided that no further analyses will be performed with that data), it can delete them. Alternately (or perhaps additionally) the system can be set up such that results sets are deleted or archived after a certain amount of time to improve system performance.

Once the CS has received all of the necessary data and sends the confirmation message to the UIS, it begins processing. As with the attribute services above, it also can send periodic messages with progress reports to the UIS. Like with the attribute services, the CS may also request equivalency details from the OEI. When its processing is complete, another message is sent to the UIS indicating such (as before, results sets are not sent automatically).

In this example, the user decides to view the results from the CS. In the unlikely case that the user would view the raw results, it might direct the CS to provide data back to the UIS directly. This is unlikely because it would leave the user examining a set of fusion results in terms of object identifiers that have no qualitative meaning to the user. More commonly, the UIS would direct the CS to send some portion of the results (perhaps the N best matches) to several different attribute services with a second type of request. Where the first request provided measurements and produced a ranked list of possible matches, this request provides object identifiers and returns measurement values.

The specific attribute services accessed at this stage are based upon the user's preference for how to view results. It might be that the user wishes to see just a name associated with the object, or perhaps a whole range of attributes (names, images, other identifiers, etc.).

As above, these attribute services alert the UIS when the data is available, and UIS now instructs the attribute services to send results sets back to the UIS. Those sets are then formatted and finally displayed to the user (using other GUIs that are provided by the respective attribute services).

This was a relatively straightforward example of the architecture information exchanges, intended to illustrate the basic concepts. Additional information exchanges can also occur, such as when one AS has measurements that are functionally related to other attributes. In those cases, additional calls are required, such as to the RS to obtain network addresses, and between attribute services to obtain the necessary information.

IV. Implementation

To illustrate the architecture, a simple prototype implementation was developed using Microsoft .NET. For clarity of concept, the prototype demonstrated correlation between two readily understood data types: person names and social security numbers. Person names were chosen because of their moderate complexity (people may have several names of various types – given names, family names, past names, aliases, etc.). Social Security Numbers were selected for demonstration purposes because the construction of a person's social security number includes information on the location where the number was issued, resulting in more complex results from this service (i.e. insight into a location that is affiliated with that person). In this example, each type of data was given its own attribute service: the *Person Name Attribute Service (PN AS)* and the *Social Security Number Attribute Service (SSN AS)*. In order to achieve more interesting statistical results, both attribute services were designed to use the Damerau-Levenshtein String Matching Algorithm [9]. If the string matched exactly, a confidence of "1" was returned. Otherwise, the Damerau-Levenshtein distance is used to compute a confidence level.

The prototype design was heavily influenced by the requirement that the UIS GUI must be able to remain unchanged as new attribute services are added. Consequently, the attribute services provide their own GUI functionality to the user through the UIS's GUI, at run-time. This is accomplished by serializing .NET assemblies and transmitting them, in binary, via .NET remoting. These assemblies carry, at a minimum, the GUIs that both accept user input (e.g., by way of a form) and provides user feedback (e.g., by way of a table of results). For example, a nuclear spectra attribute service might have a mechanism for efficiently summarizing spectra, or a dropdown menu for the user to select an isotope of interest, etc.

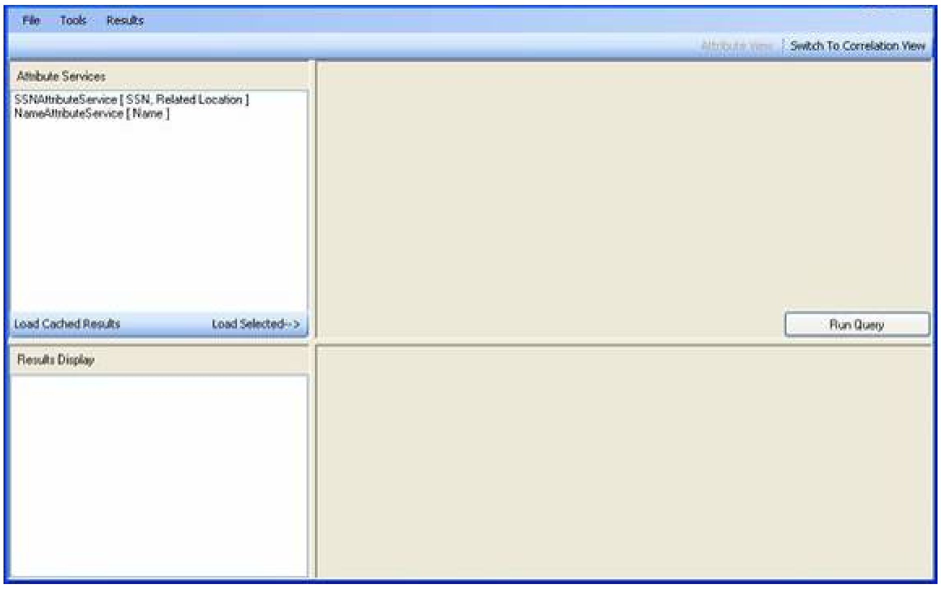
Aside from the two attribute services already mentioned, the prototype included a single CS that provided a simple linear correlation; the RS by which the other services could find each other; a basic UIS with a GUI that provides a framework for the user to view available attribute and correlation services, as well as a way to view custom graphical controls from those services; and an OI that provides a lookup of system unique identifiers to the attribute services.

The prototype uses .NET interfaces to define the contract between services. The only well-known (i.e. having a fixed port and IP address) service is the RS, the others finding each other dynamically through this service. These interfaces are:

- IAttributeService – Implemented by attribute services; includes methods for the GUI to retrieve the attribute service's assembly, and to submit the results of the form input.
- IAttributeServiceRegistrar – Implemented by the RS; includes methods for the attribute services to register themselves with the RS.
- ICorrelationService – Implemented by the correlation services; includes methods for providing data sets to the service for correlation.
- ICorrelationServiceRegistrar – Implemented by the RS; includes methods for the correlation services to register themselves with the RS.
- IPeopleServiceRegistrar – Implemented by the RS; includes methods for the OIS to register itself with the RS.

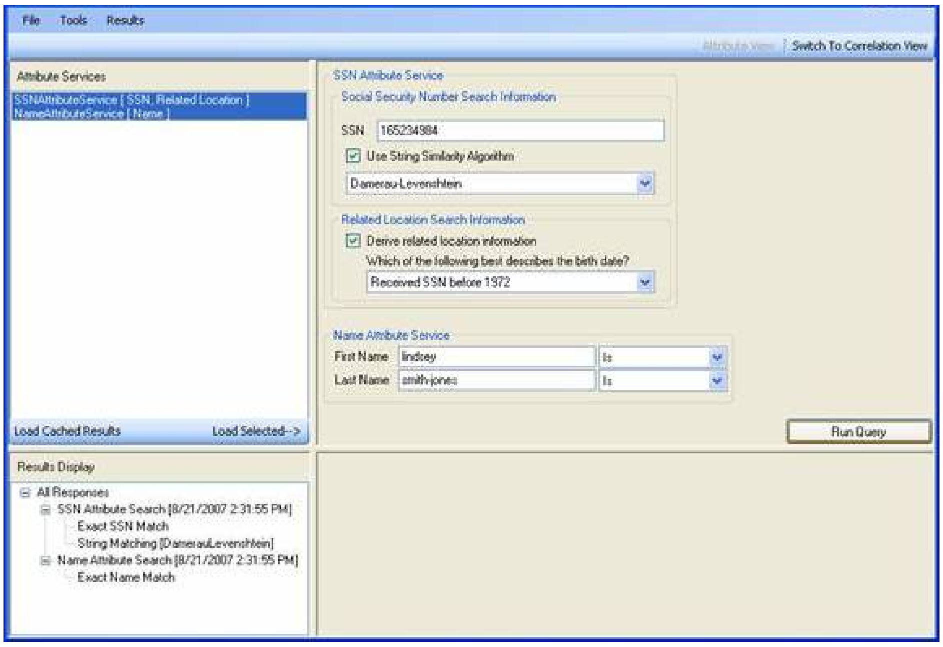
The particular interactions of these services followed the model put forth in the previous section. The user is presented with a list of available attribute services (Name and SSN), and chooses which to use. For each selected, the AS custom GUI controls are displayed, and the user fills in the appropriate data. The data are then sent to the respective attribute services for analysis. When the results are completed, the user chooses which CS to use, if any, to correlate the two results sets. In this case there is only one choice, the linear correlation service. Either or both results sets can then be sent to that CS, where they are linearly combined into a single result set. The results of the correlation service, or the original AS results sets, can then be returned directly to the user for evaluation.

Figs. 1-4 are screen captures showing these various interfaces:



SSNAttributeService [SSN, Related Location]
NameAttributeService [Name]

Fig. 1 Displaying available attribute services



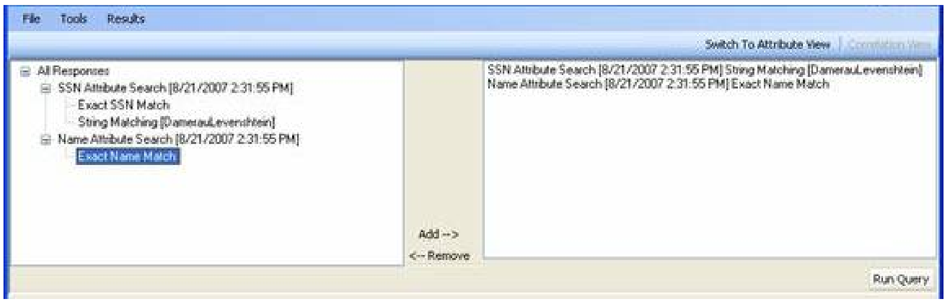
SSNAttributeService [SSN, Related Location]
NameAttributeService [Name]

SSN Attribute Service
Social Security Number Search Information
SSN 165234984
☑ Use String Similarity Algorithm
Damerau-Levenshtein
Related Location Search Information
☑ Derive related location information
Which of the following best describes the birth date?
Received SSN before 1972

Name Attribute Service
First Name lindsey Is
Last Name smith-jones Is

All Responses
SSN Attribute Search [8/21/2007 2:31:55 PM]
Exact SSN Match
String Matching [DamerauLevenshtein]
Name Attribute Search [8/21/2007 2:31:55 PM]
Exact Name Match

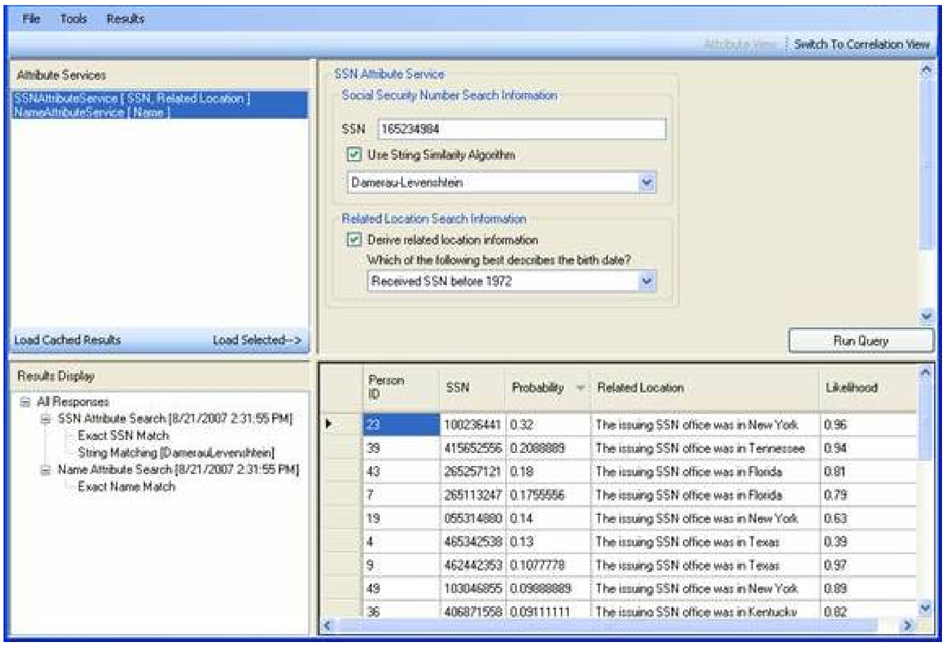
Fig. 2 Displaying GUI’s obtained from attribute services (top right panel) and existing results sets (lower left panel)



All Responses
SSN Attribute Search [8/21/2007 2:31:55 PM]
Exact SSN Match
String Matching [DamerauLevenshtein]
Name Attribute Search [8/21/2007 2:31:55 PM]
Exact Name Match

SSN Attribute Search [8/21/2007 2:31:55 PM] String Matching [DamerauLevenshtein]
Name Attribute Search [8/21/2007 2:31:55 PM] Exact Name Match

Fig. 3 Selecting results sets to send to a correlation service



Attribute Services
SSNAttributeService [SSN, Related Location]
NameAttributeService [Name]

SSN Attribute Service
Social Security Number Search Information
SSN 165234984
☑ Use String Similarity Algorithm
Damerau-Levenshtein
Related Location Search Information
☑ Derive related location information
Which of the following best describes the birth date?
Received SSN before 1972

Results Display
All Responses
SSN Attribute Search [8/21/2007 2:31:55 PM]
Exact SSN Match
String Matching [DamerauLevenshtein]
Name Attribute Search [8/21/2007 2:31:55 PM]
Exact Name Match

Person ID	SSN	Probability	Related Location	Likelihood
23	100236441	0.32	The issuing SSN office was in New York	0.96
39	415652556	0.2088889	The issuing SSN office was in Tennessee	0.94
43	265257121	0.18	The issuing SSN office was in Florida	0.81
7	265113247	0.1755556	The issuing SSN office was in Florida	0.79
19	055314880	0.14	The issuing SSN office was in New York	0.63
4	465342538	0.13	The issuing SSN office was in Texas	0.39
9	462442353	0.1077778	The issuing SSN office was in Texas	0.97
49	103046855	0.09888889	The issuing SSN office was in New York	0.89
36	406871558	0.09111111	The issuing SSN office was in Kentucky	0.82

Fig. 4 Viewing results sets (bottom left panel)

V. Summary

The architecture described herein utilizes a basic SOA approach to support both simple and complex data fusion tasks. Upstream data fusion can be implemented through dedicated attribute services that inherently address the interrelationships amongst heterogeneous data sources. Downstream fusion can be handled similarly, but also more generically through the use of either dedicated or generalized correlation services.

Because the scope and format of the data is handled through the dedicated attribute services, the architecture is extensible both in terms of the types of new data that can be included, as well as the approaches/algorithms for manipulating the data. The modularity of the individual services facilitates the continued addition of computing resources, as well as the gradual replacement of computers and servers without requiring that the system be taken offline.

The system is designed with the recognition of the importance of uncertainty in the data and the results, as well as the possibility that characteristics can apply to objects non-absolutely (i.e. in terms of both likelihood and applicability). The system handles standard data types, such as numbers, strings, dates, times, and binary objects, but also multidimensional quantities like matrices and functional representations of relationships amongst data.

Because there is significant flexibility in how a thin-client user interface can interact with the system, it is well-suited for use with business process modeling tools and approaches. This could include automating analysis processes for data flows, capturing and assessing expert knowledge and decision chains, and supporting complex data mining tasks.

In applications where long term system evolution is expected, the ability of the architecture to support ongoing development can be significant. For example, in commercial applications where data and computing resources are sold, the capability of supporting multiple competing services enables a "survival of the fittest" mechanism for system growth. Users can individually balance cost, response time, accuracy, data availability, etc., in order to optimize needs against their most significant constraints. Services that perform poorly are left behind, and the development new services that provide performance improvements are motivated.

Our experience suggests that the development costs for this type of data fusion architecture decrease over time. Initial work is required to establish the foundational services and indices (OI and RS as a minimum, and the OEI in most cases). Secondary services, such as the EI and SI, are easier to develop as they are typically variants on the more fundamental services. Similarly, the first AS and CS establish basic frameworks that, later ones can build upon speeding the development of the interfacing aspects of the newer services.

References

[1] T. Erl, "Service-Oriented Architecture: Concepts, Technology, and Design," Prentice Hall, 2005

[2] T. Erl, "SOA: Principles of Service Design," Prentice Hall, 2007

[3] http://en.wikipedia.org/wiki/Service-oriented_architecture

[4] Y. Bar-Shalom, T. E. Fortmann, "Tracking and Data Association," Academic Press, 1988

[5] "Multitarget-Multisensor Tracking," (three volumes), Y. Bar-Shalom (Ed.), Artech House

[6] H. S. Burkom, S. P. Murphy, J. S. Coberly., and K. Hurt-Mullen, "Public Health Monitoring Tools for Multiple Data Streams," Morbidity and Mortality Weekly Report (MMWR), Vol. 54 "Supplement, Syndromic Surveillance: Reports from a National Conference, 2004," 2005

[7] Z. R. Mnatsakanyan, H. S. Burkom, J. S. Coberly, and J. S. Lombardo, "Bayesian Information Fusion Networks for Biosurveillance Applications", submitted to Journal of the American Medical Informatics Association, 2007

[8] H. S. Burkom, "Biosurveillance Applying Scan Statistics with Multiple, Disparate Data Sources," Journal of Urban Health, Proceedings of the 2002 National Syndromic Surveillance Conference, Vol. 80, No. 2, Supplement 1, 2003

[9] F.J. Damerau, "A technique for computer detection and correction of spelling errors," Communications of the ACM, 1964