AFRL-IF-RS-TR-2002-142
Final Technical Report
June 2002

FUSELET DESIGN AND IMPLEMENTATION: A PRACTICAL FRAMEWORK FOR THE CREATION OF THE JOINT BATTLESPACE INFOSPHERE

Alphatech, Inc.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-142 has been reviewed and is approved for publication.

APPROVED:

JAMES R. MILLIGAN
Project Engineer

FOR THE DIRECTOR:

MICHAEL L. TALBERT
Technical Advisor
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Jun 02	Final May 01 – Apr 02

4. TITLE AND SUBTITLE
FUSELET DESIGN AND IMPLEMENTATION: A PRACTICAL FRAMEWORK FOR THE CREATION OF THE JOINT BATTLESPACE INFOSPHERE

5. FUNDING NUMBERS
C - F30602-01-C-0082
PE - 63789F
PR - 487B
TA - TD
WU - 01

6. AUTHOR(S)

Eric Jones, Basil Krikeles, Howard Reubenstein and Dan Bostwick

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Alphatech, Inc.
50 Mall Road
Burlington, MA 01803

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFTD
525 Brooks Road
Rome, NY 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2002-142

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: James R. Milligan, IFTD, 315-330-3013, milliganj@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

In this effort, two fuselet authoring tools were developed and incorporated in a demonstration system utilizing AFRL's Adaptive Sensor Fusion (ASF) technology for data communication and component encapsulation. Fuselet components created by the authoring tools fall into two categories: adaptor fuselets and correlator fuselets. These fuselets were demonstrated in the context of a Joint Battlespace Infosphere (JBI) operational scenario.

14. SUBJECT TERMS
Joint Battlespace Infosphere (JBI), fuselets, Adaptive Sensor Fusion (ASF) agents, active templates, fuselet authoring, Prolog, CORBA

15. NUMBER OF PAGES
35

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

In 1996, the Chairman of the Joint Chiefs of Staff issued Joint Vision 2010[1,2], a conceptual framework that set the stage for the Armed Services to begin to plan for future missions, and capabilities. This framework was intended as a mechanism by which the Services could leverage technological advances as a means to achieve dominance across a spectrum of operations. A cornerstone of this framework is information superiority: the ability to collect, to process, and to disseminate an uninterrupted flow of information while exploiting or denying an adversary's ability to do the same. Information superiority will be a lynchpin for the future success of joint operations.

More recently, the Air Force Scientific Advisory Board (AFSAB) has expanded and refined this concept. They recommended that the Air Force adopt as a goal the development of a Battlespace Infosphere (JBI) [3,4] a military information management system for providing integrated mission understanding, shared awareness, shared planning, shared execution, shared visualization and shared predicted views. However, in doing so, they also recognized that currently deployed military information systems are unable to meet these requirements, because they employed "closed" architectures that rely upon fixed data flows between predetermined "stovepipe" components. It is inherently difficult to modify and extend the processing configuration of these architectures, making it very difficult to meet evolving mission requirements or to adapt to a rapidly changing operational context. Typically, many months, or even years, are required to make the engineering and software modifications necessary to change the data flow of such systems. Integration of the stovepipe systems with components that were not explicitly accounted for during the design process is difficult, and often impossible.

Perhaps as important as their lack of interoperability with other military systems, today's military information systems are incapable of leveraging today's most prominent of information sources—the Internet. More often than not, this lack of interoperability with civilian information sources can be traced to either a lack of compatibility with commercial standards or the reliance upon obsolete technology. Although the recent trend in military architectural standards, for example the Joint Technical Architecture or the DII-COE standards, have been designed specifically to address these issues, significant amounts of work remain to be accomplished.

Numerous programs in recent years, including DARPA's Battlespace Awareness and Data Dissemination (BADD) Program and the Joint Task Force–Reusable Architecture ATD have started to address various aspects of this problem. The Dynamic Database (DDB) and the Dynamic Multi-User Information Fusion (DMIF) programs have focused on the problems associated with the design and construction of component-based architectures for data fusion. However, the architectures the systems developed under these programs have not adequately leveraged the most recent of commercial technologies, nor have they adequately addressed the issues associated with collecting and manipulating information in network-centric environments and the Internet.

In sharp contrast to existing "closed" architectures, the JBI must allow users to rapidly configure pre-existing software components to work together in new ways. In this effort ALPHATECH has developed fuselet authoring tools that directly address this issue.

To develop this authoring capability, ALPHATECH leveraged ongoing research efforts in DARPA's Active Templates program and an AFRL-sponsored SBIR *An Instructable Agent for*

Rapid Knowledge Base Design. Under Active Templates, ALPHATECH developed template-based restricted natural language technology for man-machine communication. Under the "Instructable Agent" SBIR, ALPHATECH is adapting this technology to support rapid specification of domain-specific ontologies, with particular attention to ontologies for processes and procedures.

We also leveraged work we have done under AFRL's Adaptive Sensor Fusion (ASF) program. ASF is a distributed systems development environment that provides a set of services to support the requirements of creating distributed systems. ASF provides a mechanism to share data based on the concept of ontologies. ASF also provides distributed component control mechanisms, point-to-point publish/subscribe being the mechanism of primary interest.

In this effort ALPHATECH has developed two fuselet authoring tools and incorporated them into a demonstration system utilizing ASF for data communications. We developed tools for authoring *Adapter Fuselets* and *Correlator Fuselets*. Adapter Fuselets are fuselets for adapting data objects of one type into data objects of another type. Correlator Fuselets detect associations among data object based on specified criteria. Our fuselet authoring tools employ graphical user interfaces (GUIs) which enable a user, who is not a trained computer programmer, to author fuselets.

We employed our fuselet authoring tools in a demonstration system to illustrate how fuselets may work in a real JBI. The demonstration system's development was driven by a use case in which the user is trying to solve the problem of getting three clients to communicate: a ground vehicle tracker, a missile launch analyzer and a visualization tool. In our use case, two fuselets are employed to facilitate the communication.

Each fuselet and client is encapsulated as an ASF component. These components use ASF publish and subscribe to communicate via CORBA. The fuselets embody a generic fuselet interface that we developed to be independent of a fuselet's functionality or underlying implementation language.

2. ASF MIDDLEWARE

2.1 ASF description and capabilities

The ASF middleware was used as a host platform for deploying the fuselets developed under this effort. In the absence of a full fledged implementation of core JBI platform services to deploy and execute our fuselets on, we used ASF as a stand-in to provide both a concrete deployment target and some services similar in nature (from a component-centric viewpoint) to those provided by the JBI core services.

A crucial design feature of the ASF architecture is the separation between lightweight, high-level ontologies for describing the battlespace and low-level data structures designed to support persistence and inter-process communication. ASF allows users to build sophisticated, flexible, high-level models of a problem domain, implemented in terms of a fixed low-level object model called the Generic Transport Object (GTO) model. Ontology definitions are treated as data (and are persisted in the data store), rather than as source code. Because ontology definitions are treated as data, the framework provides the capability to add, refine and extend these definitions without recompiling existing components or modifying the low-level database schema.

Decoupling the ontology mechanism from the transport and database data model (schema) results in a system with two desirable characteristics. First, the domain model can evolve without changing the low-level database schema. Second, the transport layer can be optimized without interfering with the domain representation.

The analogy we based our implementation on is that the JBI structured common representation was simulated via use of ASF ontology services. JBI publish/subscribe was simulated via use of ASF point-to-point publish/subscribe. We did not in any way implement a JBI core services platform, but by using mechanisms similar in nature to the services provided by the JBI platform, our fuselet experiments provide insight more readily applicable to the JBI efforts.

2.1.1 Ontologies

ASF components exchange data by making use of shared data representations defined by ontologies. An ontology is an implementation neutral definition of shared domain concepts. In an ASF-based system domain data engineering is performed by creating ontologies. The structure of ontologies is summarized in the following UML diagram.

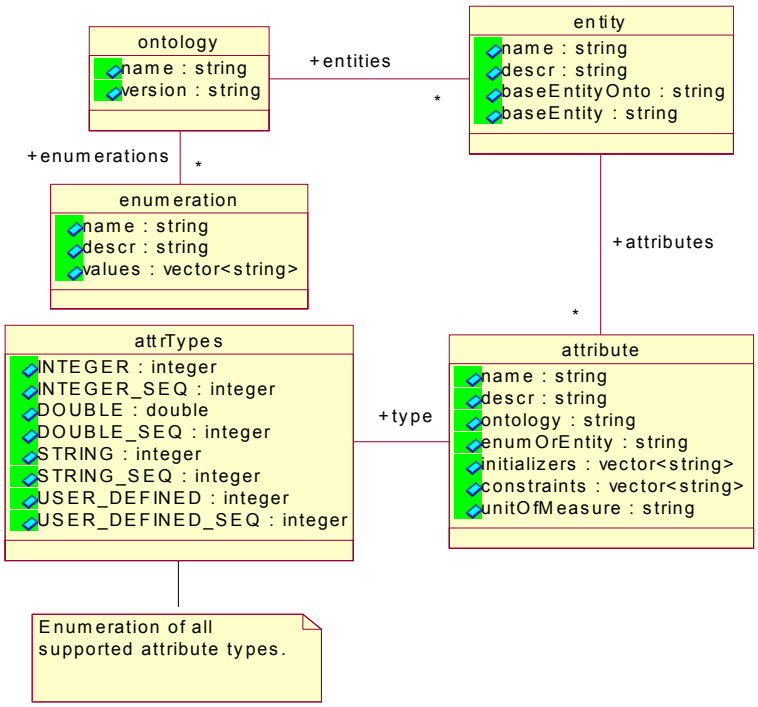

Figure 2-1 UML structure diagram of ASF ontologies

An ontology consists of a set of entity definitions with each entity having a set of typed attributes. ASF provides an ontology editor that supports the construction of ontologies. Once an ontology is defined it can be compiled into a concrete representation via the ontology compiler which will produce either Java or C++ class implementations. The generated implementations include a representation of all the domain level entities and attributes. In addition, a set of helper methods are defined that support efficient transport of ontology instances via the use of GTOs. The power of the ontology mechanisms comes from the fact that implementations are automatically generated from domain descriptions and can be regenerated after any domain engineering changes. In addition, the application programmer does not worry about the transport mechanisms as supported by GTOs. The ASF libraries and the ontology compilers provide all the code for dealing with efficient transport of data objects. The application programmer makes use of the classes generated by the ontology compiler and can use these directly in his program or can map these classes to implementation classes already present in the software module (this is part of the encapsulation methodology discussed below).

Ontology-based component development provides a methodology that allows an ASF system programmer to concentrate on the domain engineering aspects of their problem while robust and efficient implementations for use in the operational system are automatically generated by the ontology compilers.

2.1.2 Publish/Subscribe

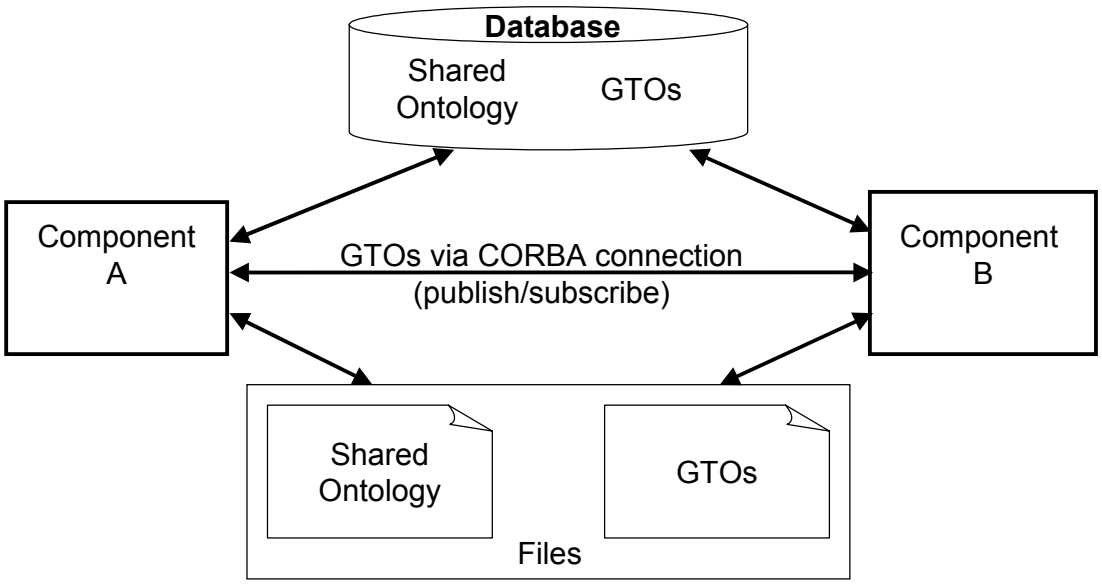

Figure 2-2 ASF Data Transport Services

ASF supports a number of data transport services all based on the GTO representation as the "wire" transport. The GTO representation provides an efficient ontology independent byte stream representation for transmission of data over persistence APIs (to files or databases) and over publish/subscribe APIs.

In this effort we make use of the ASF point-to-point publish/subscribe mechanism. Data sinks and sources can include: files, databases, or other ASF components. In this project we primarily make use of the component to component publish/subscribe mechanism and we take advantage of the file data source in our proxy components.

The ASF component to component publish/subscribe mechanism is supported by CORBA transport services. The ASF encapsulation library (describe below) supports an ASF IDL interface that captures the basic mechanisms of component to component communication and persistence interfaces. In the case of data publishing, the transport is accomplished by transport of a GTO byte sequence via a CORBA remote object method invocation.

2.1.3 Creating ASF Encapsulated Components

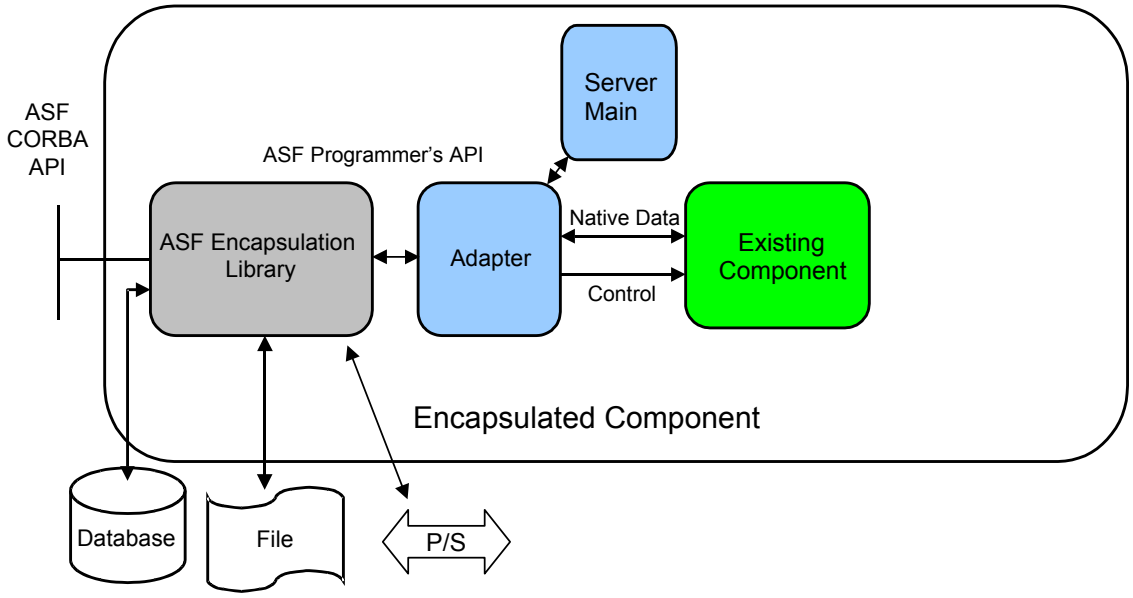

Figure 2-3 The ASF Encapsulation Methodology

Encapsulating a system (legacy or new development) to use ASF services is designed to be as unintrusive as possible to system development. The primary activity is to build a component adapter that mediates use of ASF services to the I/O needs of the encapsulated component. In addition, a main routine that launches the component and establishes CORBA connectivity (if necessary) is required. Taken together the development of the adapter and the server main() is a relatively small effort.

2.2 ASF for supporting the authoring of fuselets

ASF provides a point-to-point publish subscribe service. While point-to-point publish subscribe is fundamentally different than JBI broadcast or "anonymous" publish/subscribe, it provides a base communication mechanism that is representative of the nature of the data interchange from the components' point of view, i.e, a component subscribes to a data stream and receives input from anonymous providers.

Each full-fledged client in our demonstration system (Tracker Proxy, Launch Proxy, and Display) implements a core functionality that is middleware independent. Each of these components was encapsulated to use ASF point-to-point publish/subscribe over CORBA transport.

Each fuselet shares a common (Java) code template and is instantiated with a function specific core implementation generated by the fuselet authoring tools. The fuselet common code template makes use of the following mechanisms:

- CORBA transport via a main server routine that initializes the fuselet component and establishes connectivity with the Java ORB
- Use of ASF publish/subscribe via calls to the appropriate ASF encapsulation interfaces.

- Activation of the fuselet core implementation via use of the services provided by a generic Java-based fuselet wrapper interface. This interface provides independence from the implementation details of the fuselet computational core.
- Simple GUI component monitoring via a reusable component monitoring service.

ASF provides a computational platform for connecting the clients and fuselets in our demonstration system via mechanisms that provide JBI-like connectivity. The details of an actual JBI-based implementation would, of course, be significantly different but the overall architectural style (from a component-centric viewpoint) is similar enough to provide lessons relevant to fuselet construction and deployment in an operating JBI.

3. FUSELET AUTHORING

A fuselet authoring capability allows a user, who is not a trained computer programmer, to describe the computation to be performed by a fuselet. The fuselet authoring capability developed in this effort focused on the ability to author two types of fuselets:

- *Adapter fuselets*: Fuselets whose primary function is to supply missing "glue" code required to interconnect existing JBI clients, both pre-existing fuselets, and other JBI clients that are not fuselets. These fuselets transform the form and content of the output of one client to make it suitable for processing by another client. A simple example is a fuselet that transforms spatial information expressed in one system of geo-coordinates into another. Another, more complex example is a fuselet that extracts the data points of an MTI track and re-represents them as events for processing by an event-based fusion component.
- *Correlator fuselets*: Fuselets that perform some computation on published information to provide information of direct interest to an end-user. An example of such fuselet is a simple non-probabilistic event correlator that correlates events based upon their spatial and temporal proximity.

While we do not claim that these are the only two types of fuselets, we believe that they provide a sufficiently large space of possibilities to adequately demonstrate our fuselet authoring capability.

Our fuselet authoring tools consist of graphical user interfaces, implemented in Java, supported by XSB[5] Prolog code to implement the fuselet authoring capability. The tools represent the data types that are published and subscribed to in a JBI as classes in the ALPHATECH Knowledge Server (AKS). AKS is a lightweight frame-based knowledge representation system built on top of Prolog.

The authoring tools generate XSB Prolog code that implements the computation specified by the user. This code forms what we call the fuselet body. The fuselet body is enclosed in a Java wrapper in order to communicate via ASF as described in Section 2. The fuselet body manipulates instances in AKS; these instances correspond to the data objects in the JBI. We describe our fuselet authoring tools in more detail below.

3.1 Adapter Fuselet Authoring Tool

Adapter fuselets transform data into a form usable by other fuselets. For example, a data object may contain a number of fields irrelevant to the processing that needs to be done. One may use an adapter fuselet to create a "pared down" version of the object that contains just the fields needed for the specific data fusion application. In addition, new fields may be created which are a function of fields in the source object.

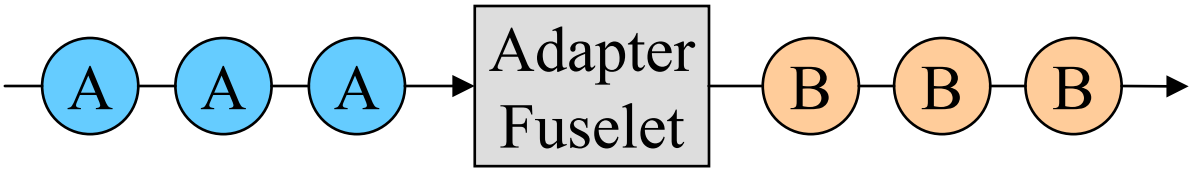

Figure 3-1 Adapter Fuselet transform data objects

As an example of how adapter fuselets are used, consider the problem of correlating track reports with launch reports. Only a few attributes of a track report are relevant for this problem, namely: the track id, track time, track status (stopped or moving), and track location. Many other attributes (such as type of sensor used, measures of error) are not used in correlating tracks with launch events. Accordingly, the track report data objects received from the GMTI tracker are transformed by an adapter fuselet into AKS objects containing just those fields.

The top-level adapter function takes an input data object and a *derived class* based on the class of the input object, and returns an instance of the derived class with field values derived from those of the source object.

To set the value of a field for a derived object from a source object, a correspondence must have been defined between the derived field and fields in the source object. The correspondence can be that of *identity* -- i.e. the value of the derived field may be identical with the value of the field of the same name in the source object. Or it may be a function of fields in the sources object -- e.g. x and y coordinates in a track point object are each functions of latitude and longitude fields in input track reports. These correspondences are generated by the Adapter Fuselet Authoring Tool and are expressed as Prolog predicates.

The Adapter Fuselet Authoring Tool window is divided into two main areas; an area showing data types available from the JBI on the left and an area showing an adapted type on the right, see Figure 3-2.

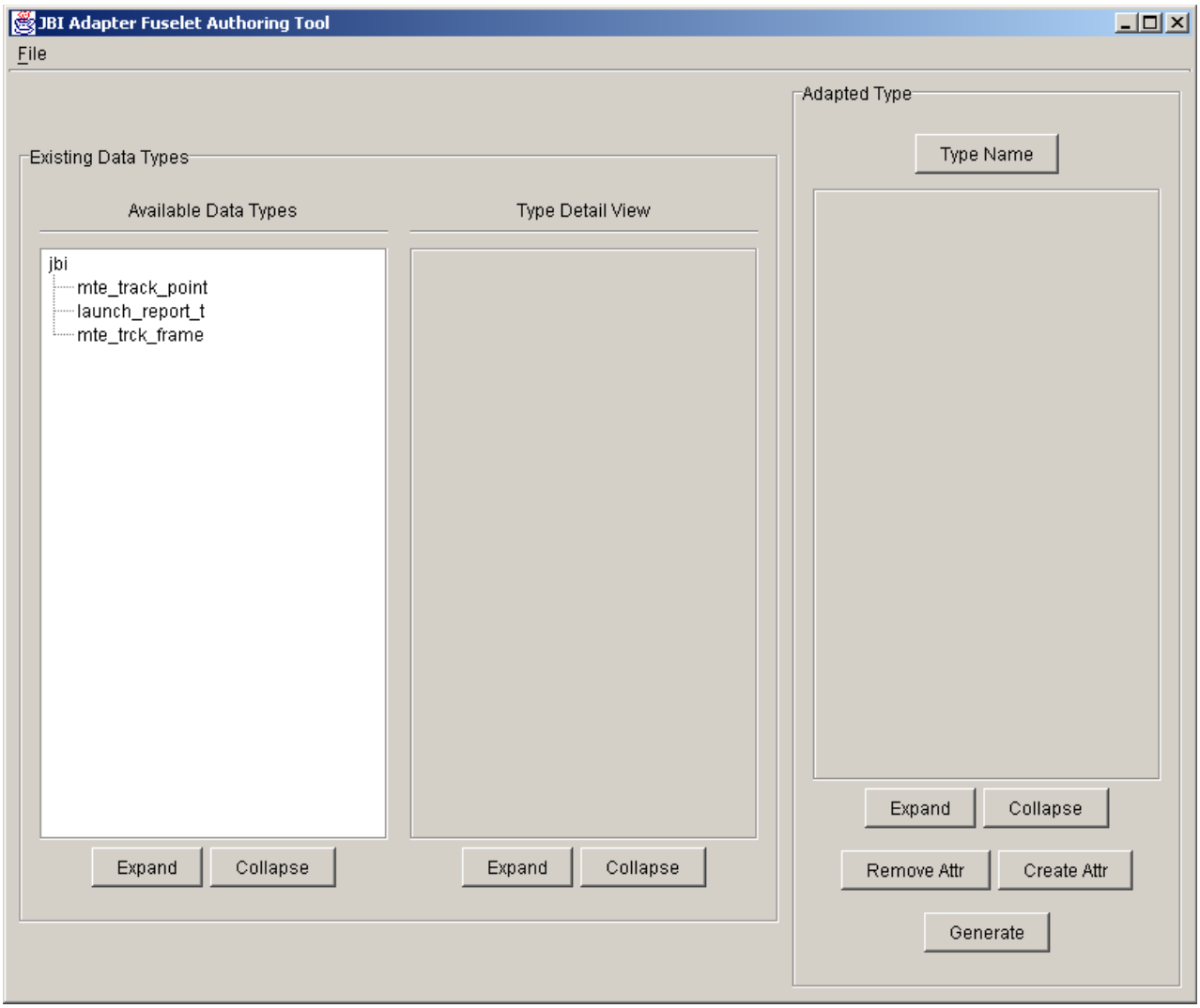

Figure 3-2 JBI Adapter Fuselet Tool

The Available Data Types panel is further subdivided into two areas; a panel showing all available types on the left and an area showing a particular data type on the right. The Available Data Types panel shows all data types that are available to adaptation. In Figure 3-2, the types *mte_track_point, mte_trck_frame* and *launch_report_t* are available. Double clicking on one of those type names will display the type in the Type Detail View panel as in Figure 3-3 where the type *mte_trck_frame* is shown.

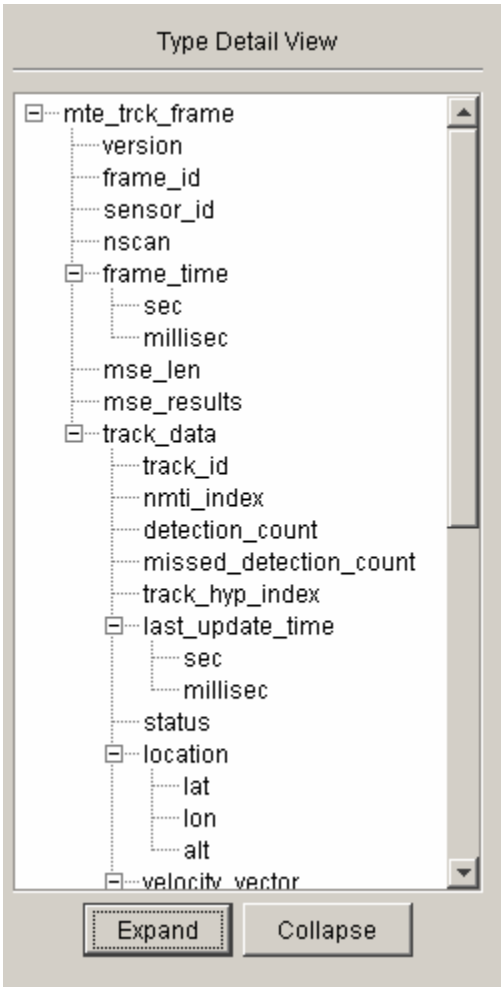

Figure 3-3 Type Detail panel

The Type Detail panel shows one particular type in a tree view. The root of the tree is the name of the type and its children are the attributes of the type. Attributes whose type is another data type may be expanded to show the attributes of that type. For example, in the type *mte_trck_frame* you can click next to the attribute *frame_time* to see that *frame_time* has two attributes of its own, *sec* and *millisec*. The buttons at the bottom of the panel may be used to fully expand or collapse the tree view.

The Adapted Type panel is where the user defines how one type is adapted from an existing type. The Type Name button is used to name the new type. In Figure 3-4, the user has named a new type *track_point*. The Adapted Type panel uses a tree view to display data types just like the Type Detail View panel. The root of the tree is the name of the adapted type and the children are the attributes of the type.

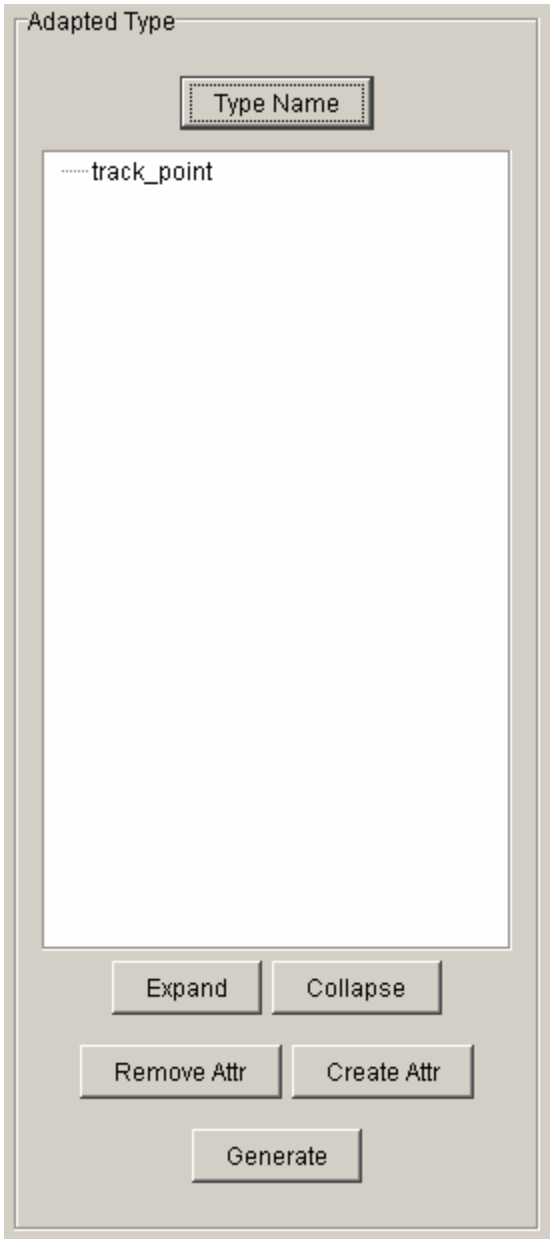

Figure 3-4 Adapted Type panel with newly named type

Attributes may be added to an adapted type in two ways; they may be directly copied from an existing type or they may be the result of applying a function to attributes of an existing type. To copy an attribute from an existing type the user simply drags an attribute from a type in the Type Detail View panel and drops it on the name of the adapted type. This is how the attributes *track_id* and *status* were added to *track_point* in Figure 3-5.

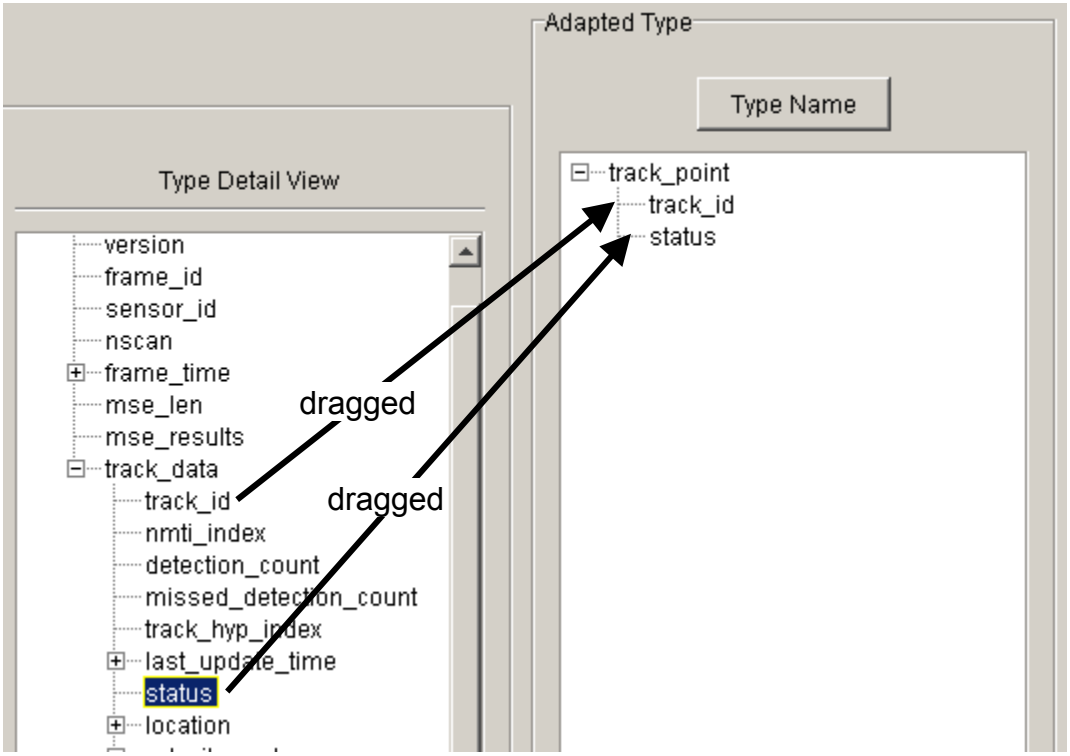

Figure 3-5 *Track_point* type with two new attributes

To define a new attribute in terms of a function of attributes of an existing type, click on the Create Attr button at the bottom of the Adapted Type Panel. This displays the Create New Attribute dialog box, Figure 3-6.

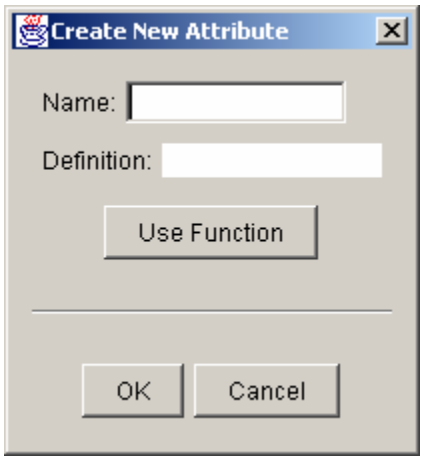

Figure 3-6 Create New Attribute dialog box

Using this dialog box, the user can name the new attribute and provide a functional definition of the new attribute. To choose a function, click on the Use Function button; a list of available functions is displayed for the user to choose, Figure 3-7.

Figure 3-7 Function selection dialog box showing two available functions

Once a function is chosen, it appears in the Definition text field with missing arguments. The user fills in the arguments by dragging attributes from a type in the Type Detail View panel into the Definition text field. Figure 3-8, shows a new attribute, *x*, that is defined as a function of the attributes *lat* and *lon*. *X* will be the x-coordinate of the conversion of (*lat*,*lon*) to UTM coordinates.

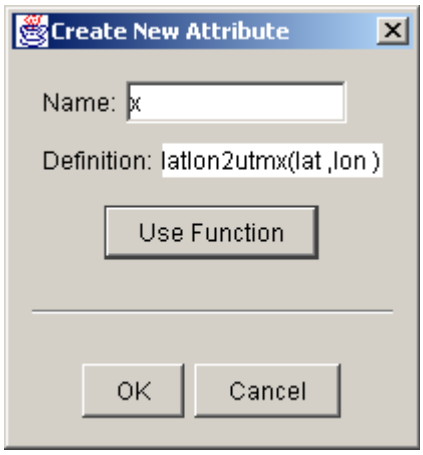

Figure 3-8 New attribute *x* defined in terms of a function of *lat* and *lon*

Clicking the OK button adds the new attribute to the new type, as shown in Figure 3-9.

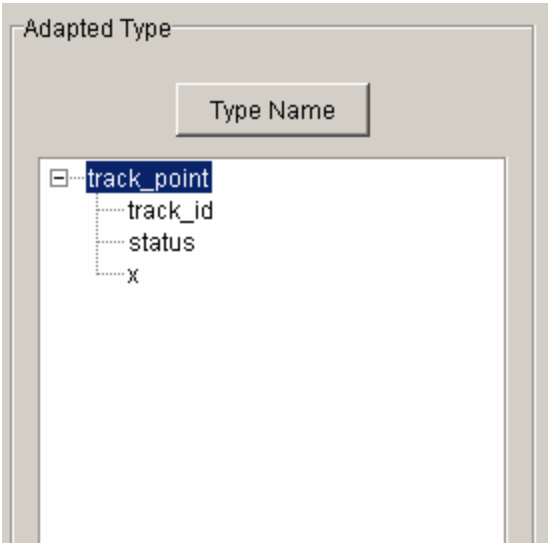

Figure 3-9 New attribute *x* added to *track_point*

The user can see how an attribute of a new type was derived by double clicking on the attribute in the Adapted Type panel, this will display a dialog box showing the origins of the attribute. In Figure 3-10, we can see that the *track_id* attribute originated from the type *mte_trck_frame*, attribute *track_data*, attribute *track_id*.

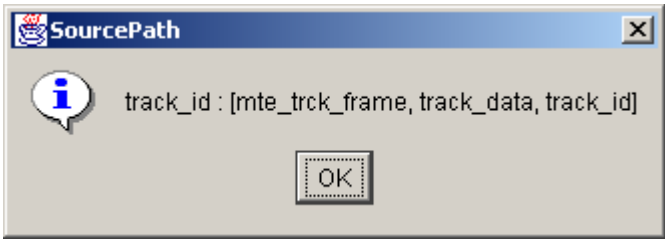

Figure 3-10 Dialog box showing the source of the attribute *track_id*

The Expand and Collapse buttons at the bottom of the Adapted Type panel may be used to fully expand or collapse the entire tree view. To remove an attribute form a new type, the user may select it and then click the Remove Attr button. When the adaptation has been fully defined, clicking the Generate button will generate the code needed to implement the adaptation.

3.2 Correlator Fuselet Authoring Tool

A correlator fuselet detects associations among data objects based on a user-defined association criterion. In our use case, the object is to correlate track reports with launch events. We wish to find start or endpoints of tracks that are within a certain distance from a launch event and whose times are sufficiently close to the time of launch.

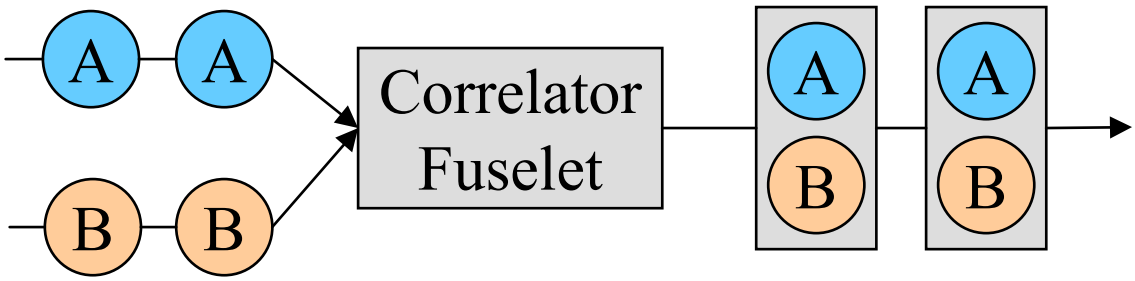

Figure 3-11 Correlator fuselets associate objects

To create such a fuselet, the fuselet author generates a fuselet script using the restricted natural language interface of the Correlator Fuselet Authoring Tool. This script is generated according to a specified *script grammar*. This grammar is used to generate a *parse tree* for the script that is input to the *script translator*. The script translator takes the parse tree and recursively translates it into executable code, Figure 3-12.

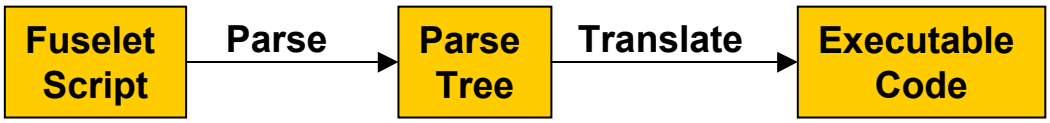

Figure 3-12 Process for generating executable code from correlation script

The following figures illustrate this process with an example. Figure 3-13 shows an example of a correlation script in English.

```
Output id of mte_track_point and id of
launch_report_t where status of
mte_track_point is equal to 4 and sec
of mte_track_point is no more than 3600
seconds before sec of launch_time_t and
the distance between mte_track_point
and launch_location_t is less than 1000
meters
```

Figure 3-13 Correlation script

Figure 3-14 shows part of the parse tree generated by the script grammar for the correlation script.

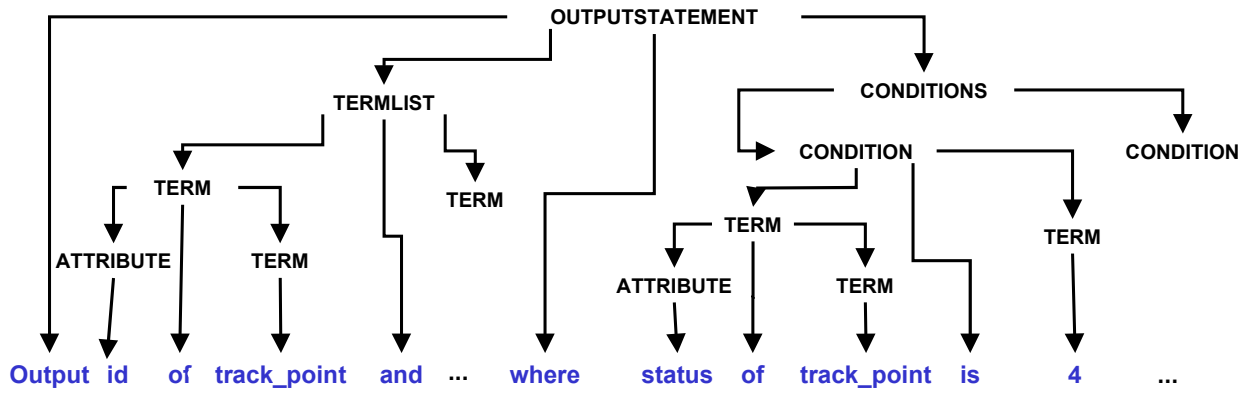

Figure 3-14 Parse tree

Figure 3-15 shows the executable Prolog code generated from the parse tree.

```
:-[quantities].
correlate(Mte_track_point,Launch_report_t,Output):-
          findall([Track_id15,Launch_id16],(
              isa(jbi,Mte_track_point,mte_track_point),
              value(jbi,Mte_track_point,track_id,Track_id15),
              isa(jbi,Launch_report_t,launch_report_t),
              value(jbi,Launch_report_t,launch_id,Launch_id16),
              value(jbi,Mte_track_point,status,Status17),
              Status17 == 4,
              value(jbi,Mte_track_point,sec,Sec18),
              value(jbi,Launch_report_t,sec,Sec19),
              noMoreThanBefore(seconds(3600))(Sec18,Sec19),
              distance_between(Mte_track_point,Launch_report_t,Distance20),
              (Distance20) =< (1000)),
              Output).
```

Figure 3-15 Executable code for correlator fuselet

The JBI Correlator Fuselet Authoring Tool allows for the creation of correlator fuselets without the need to understand computer programming. The left panel of the JBI Correlator Tool, Figure 3-16, is the same as the left panel in the Adapter Fuselet Authoring tool; it shows the data types available from the JBI and the details of one of those types.

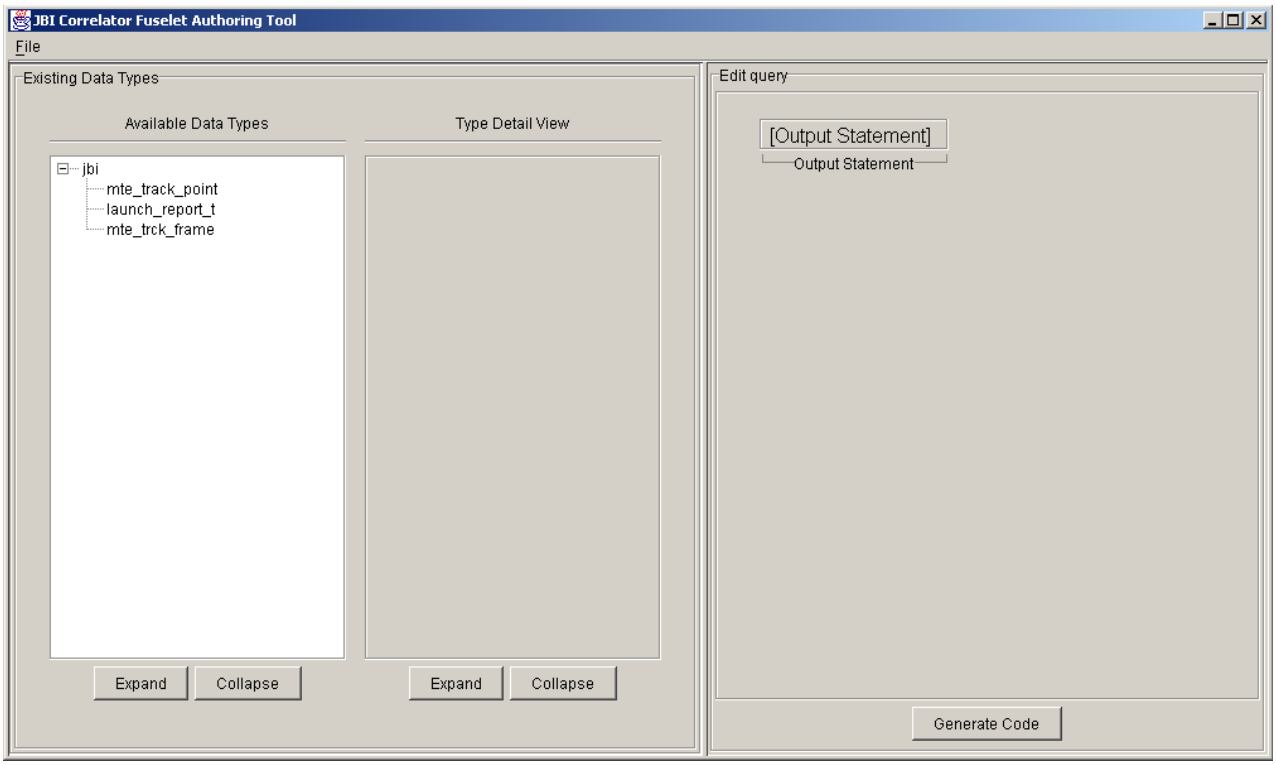

Figure 3-16 JBI Correlator Tool

The right panel of the JBI Correlator Tool, contains the Edit Query panel where users specify correlation criteria in the form of English sentences. Sentences may only describe correlations between types that are listed in the left panel. When creating sentences, type names correspond to sentence fragments labeled as "Terms" or "atomic_term". Attributes of types correspond to sentence fragments labeled as "Attribute". Users may select a type or type item in the Type Detail View and drag the item to the appropriate textbox in the Edit Query panel. The type names and type attributes are available via drop-down menus so using the drag and drop feature is not necessary if the user chooses not to use it.

Once the sentence describing the correlation has been completed, the user clicks on the Generate Code at the bottom of the panel to automatically generate code implementing the specified correlation in a correlator fuselet.

3.2.1 Correlator Natural Language Interface

The Correlator Natural Language Interface (NL interface) in the Edit Query panel of the correlator fuselet Authoring Tool is used to create sentences based on a pre-specified grammar. Using the NL Interface, correlation criteria are specified in the form of natural language text.

The NL interface is viewed as a GUI in written in Java with a back end in Prolog. The NL interface constrains the set of allowable sentences to those sentences that can be generated from a definite clause grammar. The NL interface gives the user the ability to create sentences without having to know the grammar beforehand. After a sentence has been created in the NL interface, data in the sentence can parsed by the back end for use by other software.

Combining parts of speech such as nouns, verbs, and prepositional phrases in a certain sequence creates sentences. In the same manner, sentences created with the NL interface are composed of elements in a certain sequence. In Figure 3-17 the "Output Statement" sentence is created by combining elements and required text in a specific order. The labels beneath each element indicate what type of information the element contains. There are three types of elements: required text elements, free text elements, and regular elements.

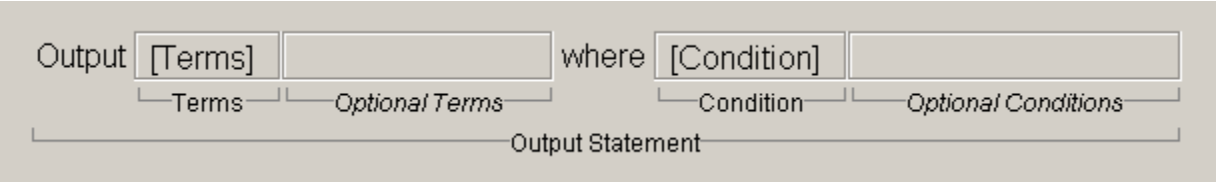

Figure 3-17 Example of a NL Interface

Required text is text that is always associated with a element. For example, the "Output Statement" sentence in Figure 3-17 always contains the required text "Output" and "where".

GUI elements represent sentence fragments that consist of sub-elements and/or required text. Figure 3-17 shows the expansion of the element "Output Statement." Users can expand a GUI element by left clicking the label beneath the element. If more than one expansion is available, as in Figure 3-18 for the "Condition" element, a pop up menu will list the available expansions. The user can choose one of these expansions simply by clicking on the desired expansion.

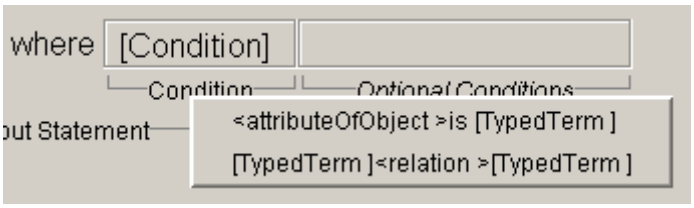

Figure 3-18 Example of pop-up selection menu

In Figure 3-18 the "attribute Of Object" and "relation" are placed within angle brackets("<>"), while "Typed Term" is placed within square brackets ("[]"). These symbols are used to denote whether a sub-element is required or optional. Sub-elements that are optional are placed within angle brackets ("<>") while sub-elements that are required are placed within square brackets ("[]"). Required text such as the "is" in Figure 3-18 are not placed within any symbols. Grammar elements that are optional also have labels which are italicized.

In Figure 3-17, "Optional Terms" and "Optional Conditions" have italicized labels because they are optional elements while the required elements, "Terms" and "Condition" have normal labels. Required elements are displayed as containing their label inside brackets to identify that the element is not yet fully specified.

To expand GUI elements, the user left clicks on the element's label. To collapse the display of a grammar element, the user right clicks on the element's label. When an element has been collapsed, it displays the sentence fragment text that was generated. If the user decides not to

keep the text in a element, the text may be deleted by right clicking on the element. This will remove the previously generated text from the element.

Some GUI elements are colored white as in Figure 3-19. These text fields allow the user to type in any text directly into the element. To delete the typed text, select it and press backspace. Unlike the gray element, labels beneath these elements have no functionality when clicked upon. Text entered into one of these free-text fields will not be parsed by the Prolog backend.

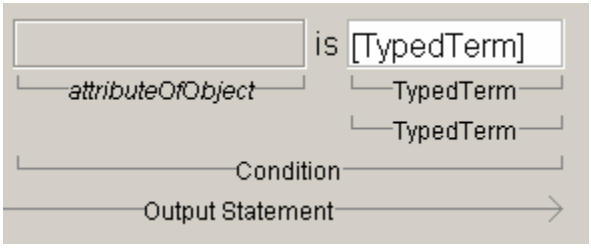

Figure 3-19 Example of Free Text

For example, information entered into the "TypedTerm" field in Figure 3-19 will be understood by the system as a condition. While it is possible to type an entire sentence into these elements, the typed text will have no meaning to the system other than the description given below the field.

4. DEMONSTRATION SYSTEM

4.1 Objective of System

The purpose of the ALPHATECH Fuselet Authoring Demonstration System is to demonstrate :

- Fuselet authoring tools that enable users who are not computer programmers to describe the computations to be carried out by JBI fuselets.

4.2 Example Problem

We developed a use case that drove progress on this effort. It provides examples of both types of fuselets described above. Our use case employs several existing software components: a (proxy for a) tracker, a (proxy for a) missile launch analyzer, and a visualization tool. Each of these components will serve as a pre-existing non-fuselet JBI client—a provider of information services within the publish/subscribe framework of JBI. The tracker provides tracks of ground vehicles; the launch analyzer provides information on the location and time of perceived missile launches; and the visualization tool displays the data on a map-based display.

In our use case, the user wishes to correlate vehicle tracks with missile launches to determine which vehicles are possibly involved in supporting missile launches. To do this requires several steps: first, appropriate information sources must be identified; second, data of the exact type needed must be extracted from these sources and (possibly) reformatted for consumption by other components; and third, the data must be combined in the appropriate way. The fuselet authoring tools will support all three steps.

When the outputs of information sources are represented in a sufficiently rich ontology, or are registered in a data products library, the fuselet authoring tools can use this representation to locate information sources from which the desired data can be extracted. The user will be able, for example, to search for components (JBI clients) or information sources that produce vehicle tracks and launch reports and a component (JBI client) that can visualize the data.

Moreover, when the output of information sources are represented as structured objects within this ontology, a fuselet can be constructed that extracts precisely the data wanted from a complex object in which the data is present in some form. For example, the GMTI tracker produces a wealth of data pertaining to a multitude of vehicle tracks. Using a formal representation of the output of the GMTI tracker, the Adapter Fuselet Authoring Tool will enable a user to select just the subset of that data which is of interest. The user will also be able to specify an appropriate re-representation of this data—in our use case, track reports expressed in latitude and longitude are transformed to UTM coordinates. The code that extracts and re-represents this data is an example of an *adapter fuselet.*

Once the user has identified the proper components and deployed adapter fuselets to compute appropriate information from the outputs of those components, the Correlator Fuselet Authoring Tool can help the user develop appropriate *correlator fuselets* that exploit this information. In our use case, the fuselet authoring tool assists the user in specifying a correlator fuselet that correlates reports of track initiation and termination with launch reports. The correlation is a relatively simple, non-probabilistic method in which the user specifies temporal and spatial regions around launch events and correlates any track endpoints that fall within those

regions. For example, the user may specify that any track termination event that is within 2 kilometers of a launch event and at least and one hour prior to the launch event should be correlated with that launch event. The Correlator Fuselet Authoring Tool then generates executable code to perform the correlation.

4.3 Scenario

For our demonstration we are using data from a scenario developed at ALPHATECH for the DMTIFE program. The scenario is based in Iran and features more than 200 tracks and six missile launches.

4.4 Components

Our demonstration system consists of the following components:

- GMTI Tracker Proxy (corresponds to an existing JBI client)
- Launch Analyzer Proxy (corresponds to an existing JBI client)
- IBVS visualization tool (corresponds to an existing JBI client)
- Track Adapter Fuselet (fuselet created by user)
- Correlator Fuselet (fuselet created by user)
- Adapter Fuselet Authoring Tool
- Correlator Fuselet Authoring Tool

The two fuselet authoring tools generate XSB Prolog code that implements the computational core of the fuselets. The remaining components demonstrate the use of the fuselets to perform data transformation and fusion tasks.

The ASF-wrapped components each display a status window indicating the number of objects received and published. The status window for the track proxy component is shown in Figure 4-1. These status windows are for illustrative purposes only, they are not vital to the operation of the components.

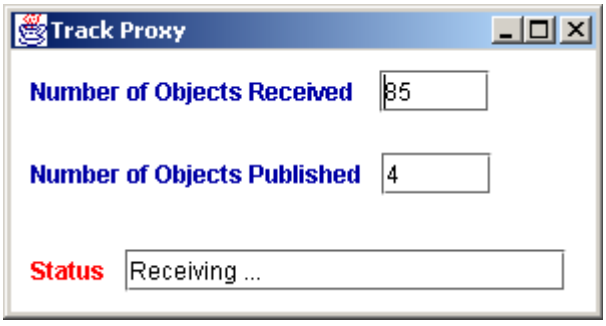

Figure 4-1 Component status window

4.4.1 Ontologies

ASF provides the concept of ontology to describe data that can be transported via publish/subscribe mechanisms or persisted via a variety of mechanisms. In the ALPHATECH Fuselet Authoring Demonstration System, we utilize four ontologies to describe the data in our system. These are:

4.4.1.1 MTETrackFrame

The objects in the MTETrackFrame ontology describe the output of a GMTI tracker. An MTETrackFrame is described below:

MTETrackFrame

- Version – version of the tracker
- Frame_id – identifier for this frame
- Sensor_id – id of sensor
- Nscan – maximum hypothesis tree depth
- Frame_time – time stamp of this frame
- MSE_length – length of automatic target recognition (ATR) results vector
- MSE_results – ATR results
- Track_Data – there is one of these for each currently active track
    - Track_id – id of the track
    - NMTI_index – index of most recent sensor report
    - Detection_count – number of frames with associated report Missed_detection_count – number of frames w/o associated report
    - Track_hypothesis_index – hypothesis index for this track
    - Last_update_time – time stamp for this track data
    - Status – status of the track
    - Location – location of the track
    - Velocity_vector – x and y velocities of the track
    - Covariance – a measure of error for this track data
    - Track_type – the track type
    - Is_group_track – whether or not this track is part of a group
    - ATR_classification – ATR class of the track
    - VLF_classification – ATR class over time
    - Onroad_data – info on road link track is on
    - Track_group_data – info on group track is part of
    - Attribute_data – size of vehicle being tracked

4.4.1.2 LaunchReport

The objects in the LaunchReport ontology describe reports of missile launches. A LaunchReport is described below:

LaunchReport

- Launch_id – id of this launch report
- Launch_time – time of this launch report
- Launch_location – location of this launch

4.4.1.3 TrackPoint

The objects in the TrackPoint ontology describe an individual track report in an MTETrackFrame. A TrackPoint is described below:

TrackPoint

- ID – the id of the track
- Status – the status of the track at this point
- Time – the timestamp on the track point
- Location – the location of the track at this point

4.4.1.4 TrackLaunchCorrelation

The objects in the TrackLaunchCorrelation ontology describe which tracks and which launches are correlated. A TrackLaunchCorrelation is described below:

TrackLaunchCorrelation

- Track_id – id of the track that is correlated
- Launch_id – id of the launch that is correlated

4.4.2 Data Sources

4.4.2.1 Launch Analyzer Proxy

The purpose of the LaunchAnalyzerProxy is to publish missile launch reports into the demonstration system. The LaunchAnalyzerProxy reads the missile launch reports from a file containing GTOs which represent LaunchReport objects from the LaunchReport ASF ontology. The LaunchAnalyzerProxy reads each GTO, examines its timestamp, and publishes it at the appropriate time.

The LaunchAnalyzerProxy can be instructed, via a Java properties file, to begin processing at a specified time step and to compress time by some factor. These parameters are provided to speed up the demonstration since the launches don't appear until timestep 1684.

4.4.2.2 GMTI Tracker Proxy

The purpose of the TrackerProxy is to publish track reports of ground vehicles into the demonstration system. The TrackerProxy reads GMTI track reports from a file containing GTOs

which represent MTETrackFrame objects from the MTETrackFrame ASF ontology. The TrackerProxy reads each GTO, examines its timestamp, and publishes it at the appropriate time.

The TrackerProxy can be instructed, via a Java properties file, to begin processing at a specified time step and to compress time by some factor. These parameters are provided for demonstration purposes as the track data contains one frame only every 20 seconds.

4.4.3 Fuselets

4.4.3.1 FuseletBody Interface

We implemented a generic ASF Java-based fuselet wrapper component that serves as a template for the fuselet implementations. The wrapper component handles registration with CORBA, supports creation of data subscriptions, establishes handlers for the data received over subscription links, and supports publishing data from the component.

The wrapper defines a Java interface called FuseletBodyInterface that abstracts from the implementation details of a particular fuselet. It does not constrain the implementation language of fuselets (which in our case is XSB Prolog). The FuseletBodyInterface is a simple interface that any fuselet can implement and easily fits into our ASF-based demonstration system. There are four methods in the FuseletBodyInterface that a fuselet must implement:

- InitializeFuseletBody() - initializes the fuselet based on the contents of a Java properties file
- ReceiveInput() - receives data from an ASF subscription and transforms it into the form required by the fuselet body
- ProcessInputData() - executes the fuselet body on the current input
- ProduceOutput() - extracts the results of computation from the fuselet body for publication

4.4.3.2 Adapter Fuselet

The AdapterFuselet communicates via ASF publish and subscribe. It subscribes to objects of one data type and publishes objects of another data type. In the demonstration system, the AdapterFuselet subscribes to MTETrackFrames and publishes MTETrackPoints. When the AdapterFuselet receives an MTETrackFrame over the subscription link, it calls a module to process the incoming data. This module calls FuseletBodyInterface.ReceiveInput(), which pushes the received object into the ALPHATECH Knowledge Server. The AdapterFuselet then calls FuseletBodyInterface.ProcessInputData(), which invokes the XSB Prolog code generated by the Adapter Fuselet Authoring Tool. Then, the AdapterFuselet calls FuseletBodyInterface.ProduceOutput() which extracts the new objects from the ALPHATECH Knowledge Server. Finally, the AdapterFuselet publishes the new objects to the rest of the system.

4.4.3.3 Correlator Fuselet

The CorrelatorFuselet communicates via ASF publish and subscribe. It subscribes to objects of differing types and publishes objects describing correlations between the input objects. In the demonstration system, the CorrelatorFuselet subscribes to LaunchReport objects and MTETrackPoint objects and publishes TrackLaunchCorrelation Objects. When the

CorrelatorFuselet receives an MTETrackPoint or LaunchReport over the subscription link, it calls one of two appropriate modules to process the received data. The appropriate data process module calls FuseletBodyInterface.ReceiveInput(), which pushes the object into the ALPHATECH Knowledge Server. The CorrelatorFuselet then calls FuseletBodyInterface.ProcessInputData(), which invokes the XSB Prolog code generated by the Correlator Fuselet Authoring Tool. Then, the CorrelatorFuselet calls FuseletBodyInterface.ProduceOutput() which extracts the new objects from the ALPHATECH Knowledge Server. Finally, the CorrelatorFuselet publishes the new objects to the rest of the system.

4.4.4 Visualization

Integrated Battlefield Visualization System (IBVS) is a visualization tool that provides an immersive 3D visualization environment for battlespace awareness, see Figure 4-2. This 3D virtual environment integrates terrain, map, imagery, and target object data into a unified operating picture for the user. The user can navigate this 3D battlespace by rotating the view around any axis and zooming in and out. IBVS was developed by ALPHATECH under an SBIR sponsored by the US Army's Communications and Electronics Command. IBVS receives its data via ASF subscription.

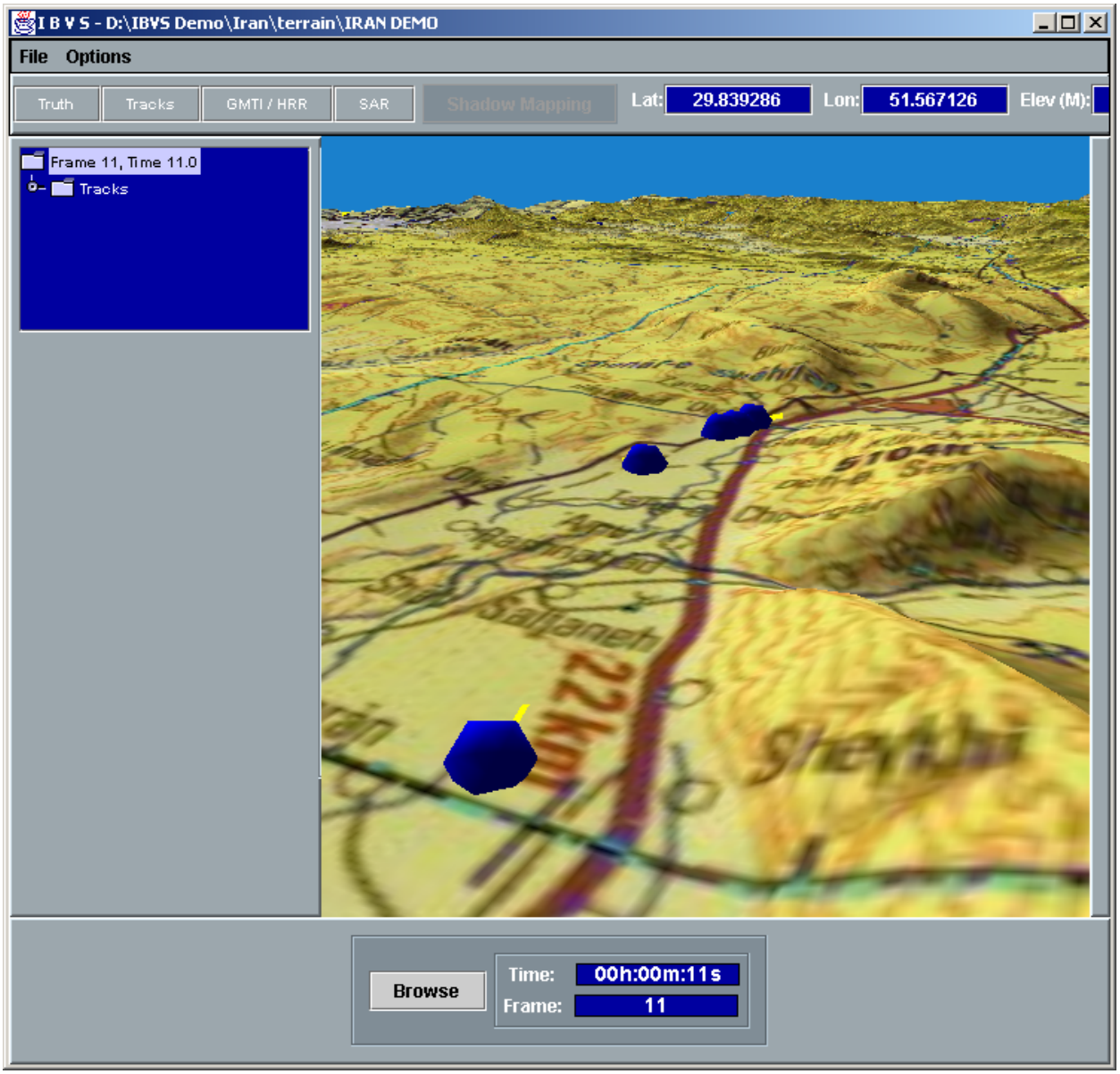

Figure 4-2 IBVS visualization tool

4.4.5 Fuselet demonstration

When the demonstration is run the following behaviors are observed. The GMTI Tracker Proxy and Launch Analyzer proxy begin publishing data. These data items are subscribed to by the adapter and correlator fuselets (and IBVS for reference in creating the display). The adapter fuselet consumes the track data and publishes track point data that is subscribed to by the correlator. The correlator merges track points and launch information and publishes launch correlations. IBVS receives track frames, launch reports, and launch correlations and uses this data to show both the launch events and a trail associated with the correlated tracks. When correlation proceeds starting from the launch event this provides a visualization of the vehicle dispersal following a launch. The

correlator can also be specified to terminate with the launch event in which case a visualization of the vehicles converging on the launch area is provided.

5. CONCLUSION

This research effort has investigated the technology needed to build tools for authoring fuselets for deployment in a JBI. We have developed two such tools and built a system to demonstrate their use.

Future work may involve developing authoring tools for other kinds of fuselets and for further developing the capabilities of ASF to support clients in a JBI.

In a fully realized JBI where clients and fuselets are built to a common architectural standard and interface providing publish/subscribe data exchange the fuselet authoring technology demonstrated in this effort will support the creation of fuselets without significant programming effort. By using publish/subscribe protocols to wire together clients and fuselets also without significant programming effort new information processing capabilities (at least those based on existing data sources) can be stood up in a JBI by operators without extensive programming skills. This technology is a significant contribution to the information superiority goals that the JBI program addresses.

REFERENCES

1. Shalikashvili, J.M., *Joint Vision 2010*, Report of the Chairman of the Joint Chiefs of Staff, 1996.

2. *Concept for Future Joint Operations: Expanding Joint Vision 2010*, May 1997.

3. *Report on Building the Joint Battlespace Infosphere (SAB-TR-99-02),* United States Air Force Scientific Advisory Board, December 17, 1999.

4. United States Air Force Scientific Advisory Board*, Report on Information Management to Support the Warrior*, SAB-TR-98-02.

5. http://xsb.sourceforge.net