Chapter 6

Situation Awareness System

Architecture

Situation awareness and related concepts defined in Chapter 2 provided characterization of information need. Chapter 3 introduced modeling strategy that is required to meet the information need. Modeling entities and process along with their characteristics with specific rules for defining attributes results in creation of a knowledge base. These comprehensive concepts are taken as basis for creating instances in given UoD. Hosting of knowledge base can provide single point of reference. From large number of concepts and instances created based on them, query mechanism is needed to retrieve required instances. For newly created instances, inferencing is employed to identify its membership to other named classes.

Among the attributes, spatial footprint is introduced as absolute spatial coordinates or an area bounded by such coordinates that occupy space in UoD. Based on the spatial footprint, the instance can be identified to stand in spatial relationship with other instances. This may have implications in information management strategy. Similar to this, the temporal footprint establishes the temporal coordinates of the instance, and in the same manner it can be established to stand in relation with itself or others. Based on identified spatial, temporal and conceptual relations of given entities, it is possible to depict situation and determine events in given UoD. Events are identified from asserted facts, as critical state change in given situation that requires some action or response from appropriate actors. The processing of assertions, detection of events, detection of actions and notification to appropriate actors requires series of information processing tasks. This task being distributed in nature, it cannot be realized with single system. Depending up on the scale of the UoD and context, multiple collaborative systems are required.
6.1 Requirements

From the discussion so far, the required solution emerges to be a system that supports modeling, instantiation, processing, and handling of information. This is done in collaborative manner where various users are contributing. This is based on existing functionality yet; present system cannot provide end user application. Hence, it is emerging as domain specific middleware that allows domain specific common functionality to the users. The awareness is provided in order to trigger tasks in various stakeholder environments.

6.1.1 Communication Requirements

In view of the above, following communications requirements are identified for situation awareness systems[89].

**Communicating Entities**  The task is to identify the participants in communication. The entity can be a human observer, a sensor or a system. In case of sensor, a sensor can report occurrence of event to an agent. The agent may react by sending command messages to actuators. From disaster management perspective, a person will communicate to another person reporting the occurrence. The receiver will consult to an agent to determine potential outcome of the event. To assess the effects of an event, the agent may notify the actors who can report status of various observables. This will again come back to the system and also trigger a job that will keep on probing the identified actors, and continue analysis and report. The storing of continuous data is performed on storage devices. The jobs and the resource also will act as source of event, informing their status. Thus both, the human actors and computer agents can be the communicating entities.

**Communication Patterns**  Communication patterns are to be realized as discussed in information processing model.

**User-System**  User-System communication takes place when user first starts the communication with the system. The determination of users’ credential and allocation of resources are performed by the system on the basis of the message content. Once user has become a member of an EOC with a specific role, he then receives control messages or Information Request messages generated by the system for further response.

**System-System**  System-System communication takes place when a new resource is joining the EOC. System then determines its membership to EOC and polls at a desired duration to
check the availability. The system may also be set-up to receive notification when specific state is achieved.

**User-User** User-User communication takes place when the explicit suggestion for taking necessary action is required. A user can be notified by the higher authorities for alteration in structure of the EOC. This User-User communication will trigger processes that will result in logical partitioning of EOC resources. Similarly direct commands issued by higher authorities can be communicated without any processing to the specific actors.

**Subscriptions** The state change may take place at the actor level, resources level or process level. When such state-change occurs, appropriate actors should be notified. The subscriber can be a person or a process. With dynamically altering situation, such alternation can be countably many. Hence, subscriptions should be appropriately managed.

**Interaction Patterns** Interaction patterns can be synchronous or asynchronous, depending upon the information requirement and type of messaging entities. Subscription to specific topics or topic tree of relevant events can be suitable in a Publish-Subscribe scenario. The control messages being exchanged amongst services may utilize message queues. The identification of appropriate interaction patterns is based on specific communication requirements of messaging entities[13].

### 6.1.2 Middleware Requirements

Middleware is defined [90] as a class of software that are employed to address complexity and heterogeneity that is characteristic to the distributed systems. It is realized as a software component providing a layer of abstraction in the middle of operating system and the client application. A particular middleware provide specific functionality commonly required by applications. In this manner, it can be seen as reusable software components that can further be utilized by application developers. With the advancement in technology the layers providing common programming abstractions have increased and middleware can be found to have many layers. Schmidt and Buschmann [91] suggested Host infrastructure Middleware, Distribution middleware, Common middleware services and domain specific middleware services as distinct layers among middleware. Such dynamic nature of the resulting system poses additional requirements on communication middleware.
**6.1 Requirements**

**Identification of Messaging Endpoints**  As situation unfolds, availability of some of the planned entities and resources may possibly be affected. In response, the volunteers and donated resources join the response work. According to the role of the voluntarily joining actors, they should be mapped either as information sources or sink. Upon detection of information requirement, these new sources can be approached with information request, as well as with other notification to adjust their response action. Likewise, newly joining computing and communication resources need to be registered in the resource pool, and should be used for collecting, processing, managing and delivering the collected information.

The challenge is to determine the limited time period for which these entities are actively involved in communication. A disastrous event may be followed by secondary events. It is also likely that while the effects of such complex events are continued, new events may take place. Hence the lifespan of each involved entities in response-work must be determined.

**Lifespan of Messaging Endpoints**  The EOC may be responding to multiple co-occurring or separate disasters overlapping in time. The time period of response to such event may overlap. In this scenario, the life-cycle of *Event Sources* and *Event Listeners* become a critical design issue.

Figure 1.3 provides a comparative view of lifespan of various concepts related to response work. During a lifespan of EOC, it is legally responsible to respond to any kind of events that may take place in the given region. For this the EOC acquires human and material resources targeted for specific type of response-work. During the lifespan of each individual planned resource, they may have to respond to multiple events. As the situation unfolds, an actual response is expected to contain dynamic set of planned and unplanned entities. The design of the communication middleware therefore must integrate life-cycle management of various messaging endpoints.

**Rules**  Rules play a vital role in determination of communicating entities. Rules are required for:

1. **Handling Event Detection**: An event can take place in various domains. Hence strategy for detection is also targeting specific features of the domain. The rules provide logical framework to determine the event detection process.

2. **Handling Actors/Resources**: The unplanned actors and resources join the EOC with no predefined agenda. In some cases, it is possible that actors/resource starts playing a specific role in the situation and that is later reported to the system. The appropriate
6.1 Requirements

role with explicit or implicit interest in the instantaneous situation should be determined using rules.

3. **Handling Subscriptions**: Identification of roles in EOC also facilitates determination of new entities as messaging endpoints. The rules can govern the management of their subscriptions in Communication Middleware.

4. **Handling Storage and Data Management**: The unit information communicated among endpoints is to be utilized to build EOC level snap-shot of the situation. A rule governed strategy to maintain the snapshot in persistent storage is needed.

5. **Managing Life Cycle of Entities**: Lifespan of messaging endpoints and related sub components should also be managed by the communication middleware with changing membership to the EOC.

**Handling User** Users are considered both-source and sink-of the EOC messaging system. Users or actors can be of two types. A person, who is assigned a specific responsibility during planning process, is generally trained and made aware of standard operating procedures with access to various resources. Thus user can be expected to be available consistently during the tenure of service. Another type of user/actor are those who volunteer their services to EOC. These volunteers may or may not be having access to resources. Their availability is generally expected to be short-lived, and hence they cannot be expected to be consistently available throughout the event response.

In such case, the EOC can have multiple programmatic abstractions of user entities as described in Table 6.1. The EOC member with specific responsibility is defined as an instance of appropriate class in the EOC Ontology. All such instances cannot assumed to be available as the discussions above. Hence the Message Oriented Middleware (MoM) should poll those instances that are absent, and those unplanned that have joined.

**Handling Data Sources** Sensors and human observations can act as data source, providing relevant information regarding the event. The programmatic abstraction of data source must

<table>
<thead>
<tr>
<th>Domain</th>
<th>Role of a User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Organization</td>
<td>Member of VO to access available resources and services and everything else that is useful in carrying out assumed role</td>
</tr>
<tr>
<td>Knowledge Base</td>
<td>Instance of specific concept(s)</td>
</tr>
<tr>
<td>Communication</td>
<td>Message Source or Sink</td>
</tr>
</tbody>
</table>

Table 6.1: User Forms
be guided by the events that are reported by the sources.

**Handling Data Resources** Data Resources provide means to retain the information regarding the situation and events in various stages of their life cycle. The data life cycle include raw information directly collected from the data sources, the cleaned state, processed state or analyzed state. What particular state during the data life cycle is to be used by eventing application is a critical design issue. The raw data are the actual facts reported by human or sensor sources, but their count and size of multiple snapshots can be difficult to handle.

**Shared Knowledgebase** The data resources can be used to store event related data in various state of their life cycle. The EOC is expected to handle all type of events that are continuously taking place. The EOC banks upon the same pool of resources and entities for in responding to various events. Thus achieving the all-hazard approach is quite difficult to capture in database schema.

**Interaction Patterns** Following issues are to be addressed to strategize the interaction among the entities:

**Managing Life Cycle of Entities** Interaction patterns that manage life cycle of messaging endpoints should support reliable messaging among services.

**Managing Processes** The appropriate pattern for communicating control messages among system should be determined by application rules. Depending upon the case FIFO or LIFO property must be appropriately selected in communication channels.

**Information Collection Pattern** The Information Collection Pattern is based on the Request-Reply pattern[13]. The modification is required in the request and response message as it is expected to handle human observation.

### 6.1.3 CASE Requirements

Chapter 4 introduced a unified process created to support information needs of the actors in carrying out assumed roles in different execution environments. Chapter 5 introduced architectural products to meet information needs of the users in software development environment by establishing coverage and gap analysis. Such requirements are generally met by class of application known as Computer Aided Software Engineering (CASE). While many popular CASE tools
provide common provisions, certain additional features must be achieved for complex dynamical systems[92].

**Supporting Separation of Concern** As discussed earlier, the actors associated in the process may have completely orthogonal concerns and hence, tooling must provide support for customized experience of each actor playing one or more roles in a given instance. This need becomes more relevant when actors just provide their skill-set and the CASE tool thereafter should be able to infer what possible roles they can play.

**Event Driven** Targeted systems are reactive in nature; and the case tool must support event-based triggers. The RUP may specify if a particular activity is event triggered or not, but the mechanism of detecting the event is not known. The events are also considered to be delivered not to the explicit subscriptions but the general roles. For example, set of trained volunteers are capable of providing many ICT related services, when specific type of event happens how they are informed for the requirement of the job. As the RUP further allows specification of skills, this information can be used for identification of the proper recipient of the event notification. Hence the CASE tool must be able to collect profiles of the potential members, and the event detection mechanism must be able to identify appropriate recipient and be able to deliver message to them over collaborative environment.

**Dynamism in Organization** In situations where organizations are responding to emergency situations or crisis, the decision for setting up an Emergency Operations Center (EOC), and decisions related to allocation of resources keep on changing as the event unfolds. As exact boundary of emergency is discovered, organizations may have to come together at federal or international level resulting in change often in drastic manners. The resources in terms of skilled manpower and ICT infrastructure may change with volunteers and donated resources that need to be incorporated instantaneously. The CASE tool must support the required level of dynamism experienced by the organizations during the events.

**Estimation of Efforts** For any organization, the volume of work should be determined. There can be some systems already in place, which can be integrated with the planned information system. The knowledge representation needs to be incorporated also in the estimation that allows the agency to identify the items they are supposed to make provision for.
Knowledge Representation  Consistent flow of information in situation awareness system is only possible if consistent representation of knowledge is adhered to throughout the process life-cycle. It also allows reusability of the domain knowledge. The spatial, temporal and semantic reasoning is very critical for the success of process, so the knowledge representation that can allow such reasoning is a basic necessity.

While knowledge representation and reasoning allows support for spatial and temporal relationship, the architectural products should be able to use this aspect. For example various grid nodes, which are part of VO, can be scattered across a larger area; so the decisions regarding data regionalization and other such provision requires spatial nature to be considered. The meta data is also having spatial elements in the schema, hence architectural product should be able to render instances and allow query based on spatial/temporal attributes.

From Monolithic to VO  The collaborating teams should be considered as members of a virtual organization (VO). Like successful applications as bug tracking is handled in collaborative mode, the process can also be handled in similar manner.

Semantics based Traceability  Traceability among architectural products provide basis for tracking the coverage of the effort. The software teams involved in SA is provided the tasks during the entire life cycle, and hence each and every part is interpreted as required due to some precursor. Thus traceability can actually be based on semantics and should be able to cover the entire set of the tasks.

Task Allotment  Task allotment to volunteers or the team members can be done based on evaluating the skills. Once some volunteer defines that they will be continuing with the task, system should be able to take notice of the same. The status of ongoing task should be identifiable at any intermediate interval.

Visualization  Complex level architectural approaches demand high degree of technical expertise for the user. How users will take up the task depends on how effectively it is provided with proper visualization. Monitoring of allotted work, overview of the process status; search of architectural components, the hierarchical view coverage and errors should be rendered to improve the quality of the development.

Standardization  The standardization related concepts in the ontology suggest their applicability. Each artifact or work product should be traced to appropriate standards. Standardization
traceability matrix not only supports the developers to consider standards, it also provides the specification that must be considered while developing given application.

**Artifact Impact and Reusability**  A true test of proposed SME in SACore can be possible at global level adoption of such methodology for responding a real world crisis situation. There are already existing global initiatives like GDACS that allows alerts at global scale, but with limited amount of flexibility for customization and to suite the individual needs of organizations.

The SA life cycle demands the knowledge of when and how reusable artifacts can be published, discovered and utilized. There can be some artifacts that are relevant for some time only for very specific region and cannot be reused. The data collected the execution environment, system level tasks, organizational decisions and reviews fall in to this category. Some artifacts are local but valid for longer period of time and do not require frequent updates. For example, SA Configuration, organizational knowledge, mapping, transformation and organizational policies. SA review and SA management can be considered to be globally relevant but short term and needs to be verified by multiple implementations. The fourth quadruple contains the set of artifacts that can be reused. For example SA Process, Services, Components Standardizations and domain knowledge, can be created or incorporated by any SA Configuration and can be reused globally for long period of time.

### 6.2 Building Blocks

Requirements identified for realizing situation awareness system indicate few recurring themes that lead to the identification of basic building blocks for the system. One such functionality is information handling in conceptual dimension. This is identified as knowledge management. Representation, processing and reasoning of the knowledge being captured by the various roles and their actions. Physical and impact footprint are two important features for entities in the UoD. Apart from conceptual dimension, the attributes in temporal and spatial dimension allow additional reasoning capability for capturing complexity and dynamism. Information handling in spatial dimension by capturing, representation, query and analysis of spatial attributes of the entity is important building block. Handling temporal dimension of information is supported by both conceptual and spatial building blocks.

Apart from processing the information, transfer of information is important recurring requirement. They are identified as various information exchange patterns. This leads to the identification of communication building block. Vast amount information handled by the system requires appropriate database management functionality that can be achieved by multiple
6.3 Design Decisions

Distributed database management is one building block that allows handling of underlying database at multiple physical instances. The handling and processing also includes features that allow analysis and reporting of continuously generated data.

Dynamic set of users, databases, required functionalities and systems that support information processing implies the requirement of computation. The computing system that allows processing of information over multiple underlying systems connected with network is required. Collectively, these dynamic sets of entities can be visualized to form a virtual organization. Handling and management of virtual organization by providing basic infrastructural services over heterogeneous components is therefore another important building block.

Figure 6.1 depicts identified building blocks. The required functionality in the discussion is depicted in the figure. Corresponding to each functionality information about realization of this functionality is achieved is also depicted as corresponding middleware. Figure 6.1 indicates appropriate middleware that fulfills the identified needs. Message Oriented Middleware (MOM)[13, 93, 12], Semantic Middleware[64], Grid Middleware[94], Data Middleware[75], and GI middleware[65, 66] are examples of popular and available middleware technologies exposing various required functionalities.

Figure 6.1: Building Blocks

6.3 Design Decisions

The identification of building blocks is an important advancement, yet it does not provide complete answer, as they must be appropriately integrated. With the integration of lower level enabling services is to be utilized to achieve higher-level domain functionalities. Hence, the integration should be based on architecture that can seamlessly interact with various services and provide the functionalities. This is achieved in the form of domain middleware components. While building these components, some design decisions are followed. Following discussion introduces some important design decision taken along with rationale for the same.
6.3 Design Decisions

6.3.1 Semantic Web Technology for Knowledge Management

The requirement of knowledge management in introduced in the beginning of Chapter 3. Entity and processes identified along with their attributes must be represented and utilized for further reference in handling situation during the runtime. The concepts may be specific to the entities given in UoD. Depending upon the requirements, entities can be modeled from various domain viewpoints. Terminology of the domain therefore also be included in the representation. The rules, standard operating procedures and other guidelines that support decision-making process are also required for determining and monitoring actions in ongoing situation. Depending upon the involvement in various activities visible in the UoD, the users can be identified to play specific roles. Various environments in which users can take actions is discussed in Chapter 4. The outcome of their actions has impact on the action of others. Hence, the entities, interactions among them and their outcomes along with processes and conditions in UoD all contribute to the requirement of handling knowledge.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Need</th>
<th>Shown as</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representation Model (Sec 3.4)</td>
<td>Create KB</td>
<td>-</td>
</tr>
<tr>
<td>Representation Model (Sec 3.4)</td>
<td>host KB</td>
<td>-</td>
</tr>
<tr>
<td>Assertion Management (Sec 3.8.1)</td>
<td>Query</td>
<td>T 7.1.1 in Figure 3.21</td>
</tr>
</tbody>
</table>

Table 6.2: Collaborative Knowledge Management Requirement

Section 3.8 introduced implementation algorithm that identifies how reference to the captured knowledge based is going to play important role in information processing. Table 6.2 enumerate a few example requirements identifying need of knowledge management activity required in proposed information management strategy.

Semantics for Knowledge Management Among other possible approaches, Semantic Web technology have proved useful for knowledge management [95]. From repository centric management of information to smaller information managing communities followed by recent trends in social computing, the knowledge content and its representation is growing at larger scales requiring dynamic view. Since its inception, the development of Semantic web technology have successfully demonstrated [96] its capabilities in meeting identified goals. This is supported by collaboration among related communities for developing theories, standards, tools and technologies that are instrumental in achieving the objectives.

Basic framework for handling semantics was already in place. Yet considering the growing information and its management, large-scale collaborations among participating communities are envisaged. One important enabling factor to support this effort is identified to standardize
6.3 Design Decisions

the handling of the knowledge being created. The Internet Engineering Task Force\(^1\) and The World Wide Web Consortium\(^2\) (W3C) standardization activities resulted in standards and specifications that enable consistent generation and handling of knowledge. Resource Description Framework (RDF) enabled triple based representation of resources exposed on the web. The growing uptake of standard resulted in repositories storing the knowledge captured in the form of triplets. This initiated technology and tooling support in handling repository. With growing needs and application scenarios, more expressive standard was required and hence the Web Ontology Language\(^97\) (OWL) is introduced. Moving from meta data handling capability of RDF, the OWL allowed creation of Ontologies meeting knowledge handling requirement of various domains. The standardization process also addresses queries. As rules are characteristic to domain knowledge provision of rule is covered in Rule Interchange Format (RIF). These standards allow knowledge representation and reasoning capability. With available formal framework of theory, standards and technology support, the collaborative effort resulted in creation and sharing of Ontologies supporting reuse of captured knowledge.

**Requirements to be met by Ontology** The development of semantic web technology can meet the identified information management requirements. The required granularity of representation can be handled by appropriate level of conceptual hierarchies captured in the ontology. Various functions requiring relation among concepts and instances is realized with semantic queries. For given instance, membership to specific class based on the predefined rules can meet the need of inference support during the runtime. All these features allow building of application that is based on ontology. Event detection for rule is employed to realized distributed event based system \(^98\). Application logic is also successfully captured using this technology. It is also employed for realizing event driven systems \(^99\).

**Alterations required in Handling Ontologies** While the practice of knowledge management is reported to be meeting the knowledge management need of real life applications, the complexity and dynamism still provide a difficult challenge. In present scenario, there can be many distinct domain of scientific and technological inquiry. Terminology, rules and other aspect of domain knowledge must be consistently captured. The allocation of task of capturing the domain knowledge and its continuous update is a difficult task allocation problem. One such approach is handling of smaller chunks of domain specific knowledge is restricted to individual ontology. And upon the requirements, they required Ontologies are integrated to meet the

---

\(^1\)IETF Web Page: http://www.ietf.org/

\(^2\)W3C Web Page: http://www.w3.org/
6.3 Design Decisions

coverage of knowledge representation. This integration is done using mapping. Some times, the concepts and not completely compatible and certain conversions are required before utilization of Ontologies. Mappings can be created and shared to support these features. Various levels of Ontologies can be identified. Based on the content and the nature of expertise in domain expert, four distinct type of ontology concepts identified [100]. Domain independent, domain specific, local specific and application specific as indicated in Figure 6.2.

The domain knowledge representation carried out by collaborating team of domain experts and knowledge engineers use desktop applications providing workbench for ontology representation. This knowledge based must be exposed in a manner that users can access it over open standards. This requires a semantic repository and that is exposed within a server based execution environment, such that the client can access it. Sesame\textsuperscript{3} [101] provides an this functionality of publishing the Ontologies and allowing access through standard semantic web query languages. SPARQL [102] is utilized for querying the published ontology.

6.3.2 Human Observation as Source of Information

Information management methodology requires instantiation, monitoring and communication of various domain concepts, resources and process as captured by the modeling process defined in Chapter 3.

\textsuperscript{3}Sesame Project: http://www.openrdf.org/
6.3 Design Decisions

<table>
<thead>
<tr>
<th>Topic</th>
<th>Need</th>
<th>Shown in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assertion Management (Sec 3.8.1)</td>
<td>Input for Algo. 3.1</td>
<td>Figure 3.21</td>
</tr>
</tbody>
</table>

Table 6.3: Information Collection Requirement

Limitations of Sensor-Only Approaches  Generally, approaches towards situation awareness mostly rely on the sensors technology [103] for their information requirements. Various sensors deployed in the area of interest plays the role of data sources that are frequently polled for tracking the change in observables. Other than the possibility to control location and adjustment in sensing frequency, these approaches do not provide the required flexibility to meet the information needs as described below.

Information Content  The information from a deployed sensor network is almost fixed. For example, a sensor network deployed for tracking the climatic condition will provide information in form of reading of particular observables like temperature, pressure, wind velocity, wind direction etc in a predefined range. Information requirements beyond this will require to set-up another set of sensors.

Information Flow  In case of sensor network, the information flow is in predefined pull or push modes. It is possible to send control messages that will change the behavior of the sensors to control the information flow. The information flow in case of EOC may vary to suit the role of actors. The human observer not only act as sink of control messages but also can act as subscriber to specific events published by other sources on the network.

Control Messages  Implementation of control message for human subscribers helps achieving the rule driven information collection that is not possible in case of sensor assemblies/sensor networks. The rules defined in ontology suggest information need for a specific situation, at a given space and time granularity. It is also possible to generate message requesting the information update that is consistent with the ontology and the rules.

Topology  The topology of a sensor network remains unchanged during the operations. The adjustments of node locations and the hierarchy are almost static. In case of command and control mechanisms, the topology and hierarchy of the information sources will be generally changed in the duration the operation persists.

In reference to the issues represented above changes required in sources of information. Human observations are appropriately handled to act as sources of information. The uncontrolled reporting from human observation may lead to semantic, syntactic heterogeneity, disambiguation and many other issues that is not suitable for the information management requirement. Hence
the strategy must be defined that allows handling of human observation to provide reporting free from these errors.

### 6.3.3 Message Queue for Processing Task Lists

Automation in the process of information management result in generation of task lists. In Section 3.8 information management strategy is introduced that employed rule driven generation of task specification for various information processing requirements. As there can be many possible information processing requirements that can be identified on the run time, these algorithms allows identification of task parameters. The identification of task and execution of task requires separate handling strategy and have unique requirements. Identification of task is triggered by some events, and is involves reference to the application rules in detecting the task parameters. Where as execution of the determined task requires conversion of an abstract task specification in to a job specification that can be submitted to job execution environment. Job specification needs to be executed at determined time along with other required inputs and arguments.

In both cases, the continuously generated specifications need to be handled appropriately. Table 6.4 indicates various task lists generated in information management strategy recommended in Chapter 3.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Form</th>
<th>As shown in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assertion Mgt. (Sec 3.8.1)</td>
<td>PrimaryInfoNeedList in Algo. (3.1)</td>
<td>L 7.1.2 in Fig. 3.21</td>
</tr>
<tr>
<td>In. Need Deter. (Sec 3.8.2 )</td>
<td>CharConceptPropList in Algo (3.2)</td>
<td>L 7.2.2 in Fig. 3.23</td>
</tr>
<tr>
<td>Mssg Pat Gen (Sec 3.8.4)</td>
<td>PatternMapList in Algo. (3.4)</td>
<td>L 7.4.1 in Fig. 3.26</td>
</tr>
</tbody>
</table>

Table 6.4: Queue Management Requirement

**Message Queue**  
Message queue is a distributed computing concept employed to efficiently process the jobs. Communication among collaborating components within distributed system is a common requirement like interaction among programs. This can be request of the task that can be responded back. There can be two possible ways of communication. When application requirement is such that calling procedure cannot execute further until is get the request served by the other component. For this it has to wait for reply thereby halts the execution. This is known as synchronous communication pattern. In many other scenarios, the requesting procedure may not require to wait and stop execution, as there is no input required from the receiver. This is known as Asynchronous communication. Message queue is example of asynchronous communication[104].
6.3 Design Decisions

Alterations required in Message Queue  Message queues are created generally with administration tool provided by the message queue technology. Some of them are created and handled programmatically. In each case, the administrator or the programmer creates and monitor the queue individually. In information management scenario in complex dynamical system, creation, monitoring and utilization of message queue should be handled automatically. Various parameters for generation of queue is determine on the runtime. This is driven by accessing the application specific rules for handling messaging patterns.

Utilization of Message Queue  All lists in Information Processing Model is considered as queue. With the help of middleware services, their administration should be appropriately handled. Also, rule driven handling of patterns should be supported at higher-level domain middleware service.

6.3.4 GML as Application Schema

Application schema is an artifact established and utilized by application developers that provides conceptual schema of data handled by the application. In information management applications where communication of data is frequently carried out among vast application user base, the XML based approach for application schema is a common practice. All the XML based interchanges are done in reference to this shared and published application schema; therefore provide consistent handling and processing of information in distributed environment.

The information management strategy recommended for situation awareness must also subscribe to appropriate schema so that consistent processing and handling of XML based exchange can be consistently handled. The requirement of application schema can be traced to the information processing models discussed in Section 3.8. Table 6.5 enumerate few instances where use of application schema is required. Encode function specified in information need specification algorithm requires the encoding of the identified information need to be done according to a suitable application schema.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Form</th>
<th>Shown in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info.Need Specification (Sec 3.8.3)</td>
<td>Encode in Algo (3.3)</td>
<td>T 7.3.7 in Fig. 3.24</td>
</tr>
<tr>
<td>Message Pattern Gen. (Sec 3.8.4)</td>
<td>Instantiate in Algo. (3.4)</td>
<td>T 7.4.3 in Fig. 3.26</td>
</tr>
<tr>
<td>ETL Sp. Gen (Sec 3.8.10)</td>
<td>GenRepTaskSpec in Algo (3.10)</td>
<td>Fig. 3.36</td>
</tr>
</tbody>
</table>

Table 6.5: Application Schema Requirement
6.3 Design Decisions

**Geographic Markup Language** Geographic Markup Language (GML) is a comprehensive specification\[105\] recommended to build application schema for handling spatial and non-spatial data. The GI applications typically require modeling, capturing, representing, processing, communication, publishing and storage of application specific features in given UoD. Many commercial and open source products handle such information in proprietary format making distributed information processing difficult challenge. The GML specification is initially devised to handle spatial and non-spatial attributes of such features within vendor neutral open source framework to address interoperability issues.

GML schema specification introduces set of XML schema that provides standard vocabulary for handling geographical objects. The base schema provides the common minimum requirement of for handling features, and other optional schema can be incorporated to suit the application need. Table 6.6 enumerates few schema introduced in GML Specification. The utilization and example is given along the name of the provided schema. With adding application specific entities along with recommended GML schema, application schema can be derived for consistent use.

<table>
<thead>
<tr>
<th>Schema</th>
<th>Use</th>
<th>Specifying</th>
</tr>
</thead>
<tbody>
<tr>
<td>gmlBase</td>
<td>base which other GML objects are derived</td>
<td>objects, properties</td>
</tr>
<tr>
<td>feature</td>
<td>representing an object from UoD</td>
<td>objects, properties</td>
</tr>
<tr>
<td>coverage and grid</td>
<td>map spatiotemporal domain to attribute values</td>
<td>spatial data structure</td>
</tr>
<tr>
<td>dataQuality</td>
<td>Reporting accuracy or precision</td>
<td>Positional Accuracy</td>
</tr>
<tr>
<td>datums</td>
<td>relation of coordinate system to a reference</td>
<td>geodetic, temporal</td>
</tr>
<tr>
<td>direction</td>
<td>Describing Directions</td>
<td>Compass Point like SW</td>
</tr>
<tr>
<td>dynamicFeature</td>
<td>Representing Time varying properties</td>
<td>time slice, history, track</td>
</tr>
<tr>
<td>geometryPrimitives</td>
<td>Primitive geometric objects</td>
<td>point, curve, surface</td>
</tr>
<tr>
<td>coordinateSystems</td>
<td>Mapping of coordinate with invariant properties</td>
<td>Cartesian, temporal, polar</td>
</tr>
<tr>
<td>observation</td>
<td>Modeling observation activity</td>
<td>Sensor, Target, result</td>
</tr>
<tr>
<td>measures</td>
<td>Specifying value of quantity with units</td>
<td>measurement</td>
</tr>
<tr>
<td>temporal</td>
<td>absolute, duration or ordinal temporal reference</td>
<td>Calendar, Clock</td>
</tr>
<tr>
<td>topology</td>
<td>representing spatial relationship</td>
<td>Node, Edge, Face</td>
</tr>
<tr>
<td>units</td>
<td>Units of Measure</td>
<td>Base Unit, Derived Unit</td>
</tr>
</tbody>
</table>

Table 6.6: Some GML Schema

**Requirements met by GML** As established by the list of schema and their utilization, the GML provide comprehensive standard based framework for handling entities in UoD. The entity model introduced in Section 3.2 required handling of footprints. These footprints are spatial, temporal and conceptual coverage of the given entity. GML is schema supports handling of spatial and temporal attributes. Apart from this, observation and measurement of instance are important aspects. Observation and measurement is also covered in the specification. Capturing
6.3 Design Decisions

dynamic features like event, tracking of object state, maintaining history etc. is also supported by the proposed schema.

Alterations required in Standard GML Handling  While, GML specification comprehensively addresses the modeling concerns of potential applications, generally the utilization is limited to the basic elements. Generally in practice, the spatial data is created in proprietary form and later exported to GML format or hosted on standard services where it is handled in GML format. The alternation is required in practice to incorporate all applicable attributes offered by the GML specification. Secondly, GML creation is to be carried out not as export format but should be included in information processing strategies. For example, in case like **Encode** function defined in Algorithm 3.3 should be encoded in reference to GML based application schema. Similarly GML is also appropriate for creating **Data Type Channel** patterns that deals with spatio-temporal attribute collection from human and sensor observations. For handling aggregation in ETL, GML is useful means for storing intermediate and final representations.

Utilization of GML  The proposed information management strategy relies on human observation as source of information. In this scenario, various inputs and observations are required from users, that are further processed and provided to appropriate users to aid their decision making process. Thus, collection of observed information must be handled consistently. The entities and processes holds attributes in spatial, temporal and semantic domain. Appropriate values should be supplied while creating a new instance or reporting monitoring status. These attributes are further processes for analysis hence should be consistently carried out. The Time coordinate should be collected with conformance to GML schema. Similarly reference to spatial location should be done appropriately. The observation and measurement also must be carried out as identified by the system. Apart from information collection, GML schema is utilized in handling messages and messaging patterns among collaborating services.

The GML specification is useful in creating information collection templates. GML BasicTypes schema introduces **gml:NullType**. It introduces a content model to indicated an absent value with one of the explanation like: inapplicable, missing, template, unknown and withheld. A template indicate value that will appear later. The information need specification indicates missing value of required object property. This should be observed and reported back to the system. From schema listed in Table 6.6 indicates possible candidates that can be useful in this case. The specification that represent part of a UoD is created as a **Coverage**. **gml:rangeSet** and **gml:domainSet** enable assignment of possible value range to specific domain point corre-
6.3 Design Decisions

Domain Range Representation in GML

```xml
<ReportClimateTemp>
  <gml:domainSet>
    <gml:MultiPoint srsName="...">
      <gml:PointMember>
        <gml:Point>
          <gml:pos>12.640314 92.76855</gml:pos>
        </gml:Point>
      </gml:PointMember>
    </gml:MultiPoint>
  </gml:domainSet>
  <gml:rangeSet>
    <gml:ValueArray>
      <gml:ValueComponents>
        <Temperature uom="urn:...">27</Temperature>
        <Temperature uom="urn:...">28</Temperature>
        <Temperature uom="urn:...">29</Temperature>
      </gml:ValueComponents>
    </gml:ValueArray>
  </gml:rangeSet>
</ReportClimateTemp>
```

Figure 6.3: Domain Range Template in GML

sponding to the UoD. The attribute value is also annotated with unit of measure. Thus, GML have sufficient provisions to enable information collection from human or sensor sources. Figure 6.3 represent an abstract representation of such template.

6.3.5 SOA as the distributed computing architecture

The nature of collaboration involves multiple systems spread across larger area. Each system may have unique specification, and hence, appropriate distributed computing strategy and information processing paradigm needs to be adopted to realize the required functionality over them. Sharing of data, functionality and capabilities are possible to achieve among systems.

**SOA** Increasing complexity in handling information in organization makes it a complex problem. It is recognized that conversion of large problem in to smaller manageable units separating the concern of the related entities provides enhanced capability of solving problem. Each unit identified corresponds to a smaller problem, is addressed with a logic easily defined, realized, executed and managed. From an organization’s perspective, organizations try to decompose and expose these small executable components addressing unit functionality. From a single users perspective, user can utilize these units to use fulfill the required needs. When this principle is followed in architecting the system, the approach is known as Service Oriented Architecture (SOA).

These composable units are known as service [106] and exhibit following features. They demonstrate loose coupling, service contract, anatomy, abstraction, reusability, composability,
discoverability and statelessness.

SOA introduces the concept of web service [107], [108]. The web service framework is characterized. Core building blocks of the architecture are Web services, service descriptions, and messages. The functionalities exposed by web services is discovered and utilized based on service descriptions provided in the standard format.

Standards and protocol are central to the web service architecture. The XML technology provides data representation and management features. The first generation of web service architecture was built using three core standards matching core requirements identified by the building blocks. For description of developed web service, Web Service Description Language (WSDL) is defined. Simple Object Access Protocol (SOAP) is developed to support message exchange pattern required in accessing the services. Universal Discovery Description and Integration (UDDI) is required as registry of available services. In its second generation, more specification were developed meeting the specific needs. The consensus driven Standardization process carried out by consortia involved participation of vendors, technology providers, research and governmental organization. Organization for the Advancement of Structured Information Standards\(^4\) (OASIS). These standards are based on web services, and hence they are benefiting from The Internet Engineering Task Force\(^5\) and The World Wide Web Consortium\(^6\) (W3C) hosted standardization process that have resulted multiple standards and specification. Each are based on the core standards and addresses specific aspect of web services upper level of the business logic. Arrangement of the standards organized according to their level and functionality is known as web service stack, representing available specifications [109, 110].

Data exposed with web service is particularly useful in disaster management scenario. The disaster alert services are capable to pin point the exact location of disasters along with the magnitude. This information must be utilized along with the domain specific rules that allow identification of impact footprint of the event. Based on the estimated spatial impact footprint the corresponding administrative blocks are identifies, that can further lead to identification or governmental organizations and officials that are legally responsible for response. The explanation provided a rule driven composition of web services to meet the user information requirements. Use of sensor web enablement is demonstrated in [111]. Orchestration of more web services is done with Business related standards in Web Service stack[112].

\(^4\)OASIS Web Page: http://www.oasis-open.org
\(^5\)IETF Web Page: http://www.ietf.org/
\(^6\)W3C Web Page: http://www.w3.org/
6.3 Design Decisions

**Alterations required in using SOA**  As discussed in the principle, SOA recommends decomposing larger problems into smaller one and exposing the functionalities over standard, vendor neutral platform such that it can be composed and consumed to suit the requirement. For organization, exposing identified and known service is relatively easy task as technology and tooling support along with organizational knowledge helps realizing this task. Yet, from user perspective, the discovery of appropriate service and utilization to meet the processing need can be a difficult challenge based on quality and quantity of candidate services available. A more difficult problem arises to the organization whose goal is in ensuring that user needs are appropriately met by services exposed by organizations. Hence Analysis and design in SOA for them must include the dynamic set of users and their standing and potential future requirements. Addressing this issue, also help increase the unplanned reuse of service. The Situation Awareness Architectural Framework recommended in Chapter 5, address this issue and provide creation of architectural products that establishes coverage and gap analysis for service.

### 6.3.6 Grid for Building Virtual organization

Complex dynamical system is characterized with a dynamic set of entities playing various roles in given UoD. Their goal-oriented nature allows them to be considered for an organizational structure. Information management strategy is focused on this organization that is targeted at provision of information. This is to be supported by various systems used by the members. Information management strategy must therefore handle information over these collaborating systems. This nature of organization is identified as Virtual Organization.

Resource sharing and information processing introduces specific challenges in VO environment. Foster et al. [113] introduced specific issues regarding resource sharing. The availability of shared resources is subject to constrain identified deliberately or due the nature and characteristic of the resource. The sharing policies may result in dynamic changes in availability of shared resource, permissions and those who can access them. This requires continuous identification, characterization and monitoring of shared resource on the run time. Depending up on nature of functionality required by a VO member, it can be supported by multiple resources or services each may be hosted by unique providers having unique sharing policies. Hence, appropriate mechanism for coordinating required operations served by multiple providers must be realized. The same resource can be utilized for multiple operations within the purview of the sharing policies that govern the access.
6.3 Design Decisions

<table>
<thead>
<tr>
<th>Topic</th>
<th>Form</th>
<th>Shown in</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETL Sp. Gen (Sec 3.8.10)</td>
<td>GenRepTaskSpec in Algo (3.10)</td>
<td>Fig. 3.36</td>
</tr>
<tr>
<td>Data Prov. Gen (Sec 3.8.12)</td>
<td>Generate in Algo (3.12)</td>
<td>Fig. 3.38</td>
</tr>
<tr>
<td>DDM Gen (Sec 3.8.13)</td>
<td>Encode in Algo (3.13)</td>
<td>Fig. 3.40</td>
</tr>
</tbody>
</table>

Table 6.7: VO Resource Management Requirement

Grid Computing  Grid computing have evolved as distributed computing platform suitable for meeting computational need of virtual organizations (VO)[110, 114]. Characteristics of grid[115] include collaboration, aggregation, virtualization, service orientation, heterogeneity, decentralized control, interoperability achieved by standardization, access transparency, scalability, reconfigurability and security. The definitions and characteristics introduced here reveals basic feature of a typical grid implementation. To aid the process of determining weather the given implementation qualifies as a grid, a three point checklist[115] is suggested by Foster. A grid coordinates resources that are not subjected to centralized control. A grid utilizes standard, open, general purpose protocols and interfaces to realize functionalities. And, a grid delivers non-trivial quality of service within service-oriented architecture.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Role</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Applications and Portals</td>
<td>Scientific, Engineering, Collaboration</td>
</tr>
<tr>
<td>User Middleware</td>
<td>Development Env./Tools</td>
<td>language, library, debugger, monitor, webtools</td>
</tr>
<tr>
<td>Core middleware</td>
<td>Dist. resource coupling</td>
<td>Security, data, process, trading, QoS</td>
</tr>
<tr>
<td>Fabric</td>
<td>Local resource manager</td>
<td>OS, queuing systems, Libraries, protocols</td>
</tr>
<tr>
<td></td>
<td>Networked resources</td>
<td>Computers, Storage systems, Networks, Data sources, Scientific instruments, Sensors</td>
</tr>
</tbody>
</table>

Table 6.8: Grid Architecture Model [114]

The Open Grid Service Architecture [116] provides set of grid middleware services that are specified as standard set of services to be supported by planned grid middleware technologies[94]. The Open Grid Forum (OGF) is currently developing grid related standards created as a merger of the Global Grid Forum (GGF) and the Enterprise Grid Alliance (EGA) that independently involved in the advancement process.

Alterations required in Managing Grid Resources  As indicated in The Grid Architecture, the grid service stack exposes various functionalities that can be consumed as service to address computing and information management needs. Rules play a vital role in expressing and utilizing complex business, application or domain logic. Hence, utilization and handling of grid resources should be driven by rules as supported by the semantic web technology. For instance, grid exposes computation power to the users. In order to tap this power appropriate
job specifications must be created and submitted to job management component of the grid. Through command line tools, desktop clients or portals, job is submitted. Computational requirements identified in the form of task specifications should be automatically converted in to job specification and it should be appropriately monitored and controlled using various grid features.

**Utilization of Grid Capability**  Dynamic set of users and resources identified characteristic to the complex dynamical systems are considered to form a Virtual Organization. Hence, Grid architecture proposed to meet distributed computing requirements of VO is utilized to set up VO for collaborating organizations. Each available member is created and handled as a member of VO. Shared services and resources following the policies are appropriately described while incorporating in the grid. The grid security is setup to handle VO level security. Computing power of grid is exposed by allowing job execution environment. Grid container is set up for hosting grid services that can be dynamically composed and consumed as application workflows[117]. Grid applications are utilized to aid specific information processing requirements.

### 6.3.7 OGSA-DAI for ETL

For information management across distributed users, Extract Transform and Load (ETL) is an important feature in controlling the flow of information among them.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Form</th>
<th>Shown in</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETL Sp. Gen</td>
<td>GenRepTaskSpec in Algo</td>
<td>Fig. 3.36</td>
</tr>
<tr>
<td>Data Prov. Gen</td>
<td>Generate in Algo</td>
<td>Fig. 3.38</td>
</tr>
<tr>
<td>DDM Gen</td>
<td>Encode in Algo</td>
<td>Fig. 3.40</td>
</tr>
</tbody>
</table>

Table 6.9: ETL Management Requirement

**OGSA-DAI**  The OGSA-DAI project [118] is involved in realizing data services as identified in OGSA Specification. It enables a grid application targeted at distributed data management. Management of distributed data in grid environment introduces special requirements. It is deployed as a service in grid execution environment. The present OGSA-DAI can be deployed on Globus container as well as the OMII container introduced in earlier. Figure 6.4 indicates the same.

RDF-OGSA-DAI 7 is the project developed to handle Ontologies[119]. The present ver-
6.3 Design Decisions

Design Decision is supporting ontology exposed with Jena2 and Sesame technology. Figure 6.4 represents components of OGSA-DAI. The following is the brief overview of main components.

**Data Resources** Data Resources can be any externally managed database system. OGSA-DAI supports, major RDBMS, XML databases and file storages.

**Data Service** Data Service provides interface to access zero or many data resources.

**Activity** Activity is unit of work. There are three type of activities supported namely: Statement Activity, Translation Activity and Delivery Activity. Statement Activities are set of SQL or XQuery commands that are supported for common functions. Translation activities are set of activities related to changing the normal data to compressed formats. Delivery Activity is set of activities that allow unique features that support delivery of the data to third party location. These locations may be a URL, Mail, FTP, GridFTP or any other valid location in Grid.

**Task** Task is set of activities with execution parameter.

**Task and Data Document** TADD contains data flow, control flow various input, output parameters, and other related information to execute the task in required format.

**Session** Management of session and transaction are two important features provided by OGSA-DAI execution engine.

Figure 6.4: OGSA-DAI Overview
6.3 Design Decisions

**Transaction**  WS-AT is supported as standard to achieve transaction in Web Service environment.

**Data Registry Service**  Data registries hold DS related information and allows discovery of DS.

**Requirements met by OGSA-DAI**  One of the basic utility of OGSA-DAI project is to include new data resources and expose it using the web service. This is useful in making the newly acquired, allocated or donated resources to be added to the service so that it can utilized in data management tasks. Present version supports various relational and XML based database technologies. With RDF-OGSA-DAI, utilized on top of OGSA-DAI instance, the shared Ontologies can also be exposed for various semantic operations.

<table>
<thead>
<tr>
<th>Type of Activity</th>
<th>Example</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Database</td>
<td>gzipCompression, gzipArchive, xsiTransform, stringTokenizer, blockAggregator</td>
<td>ogsa-dai</td>
</tr>
<tr>
<td>Transform</td>
<td>deliverToResponse, deliverToURL, deliverFromGFTP, deliverToStream, deliverToGDT, deliverFromGDT, inputStream, outputStream</td>
<td>ogsa-dai</td>
</tr>
<tr>
<td>Delivery</td>
<td>dataStore</td>
<td>ogsa-dai</td>
</tr>
<tr>
<td>Utility</td>
<td>dataSetManagementActivity, GraphManagementActivity, sparqlQueryStatementActivity, OntologyReasonerActivity</td>
<td>rdf-ogsadai</td>
</tr>
</tbody>
</table>

Table 6.10: OGSA-DAI Activity List

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- (c) International Business Machines Corporation, 2002 - 2005.-->
<!-- (c) University of Edinburgh, 2002 - 2005.-->
<!-- See OGSA-DAI-Licence.txt for licencing information.-->

<perform xmlns="http://ogsadai.org.uk/namespaces/2005/10/types">
  <documentation>
    SPARQL Desribe Query
  </documentation>

  <sparqlQueryStatement name="SPARQLDesribeQueryActivity">
    <query-request>
      <query>
        PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt;
        PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt;
        PREFIX sa: &lt;http://www.da-iict.org/research/processes.ont#&gt;

        DESCRIBE *
        WHERE { ?x ?y "Basic-phone-service"@en}"
```
Alteration required in employing OGSA-DAI  The example perform statement is represented in the code snippet is created manually. Due to the dynamic nature of the resources, static representations are not useful to act as ETL. This resource includes the source and target of the information, the granularity of the events, the availability of intermediate data sets and the logic for transformation may change during the runtime. Hence it should be generated automatically. In Chapter 3, Algorithm 3.10 represents one strategy for dynamic rule driven generation of the ETL specification that are programmatically transformed in to OGSA-DAI perform documents and executed with the service client. Similarly data provenance[120] is also required to be handled automatically with every state change experienced by the datasets.

6.3.8 Eclipse as a Tool Integration Platform

Software development platform plays important role by providing basic services to aid the software development life cycle. It provides Integrated Development Environment (IDE) for development deployment and testing of the applications using specific technologies. This includes basic programming and productivity related features like editor, debugger, compiler and deployer. Apart from this, it can handle other feature of CASE tools like requirement management, configuration management, versioning, and test management. It provides access to help and support. To support collaboration, it provides communication features. The platform must expose middleware level services at lower levels. It should support creation of rich client application.

Eclipse Project  Eclipse in its initial phase was introduced as an IDE. It is following plug-in architecture based on the standard specification[121]. In its core, eclipse has a small footprint kernel that is acting as a plug-in loader. All the functionality planned by eclipse product is realized in form of plug-ins. At the time of loading, required plug-in are identified and loaded to serve the functionality. In the plug-in based approach, a small functionality is realized and provided as a plug-in. Apart from the normal use by loading the plug-in and accessing the functionality, the declarative approach of Plug-in architecture allows description of plug-in extensions and extension points [122]. These extension points tell how the functionality rendered
6.3 Design Decisions

Figure 6.5: Generic Eclipse Plug-in Template

by the plug in can be extended to provide more complex, extended features to meet the user need. In this manner, the vast community effort resulted in developing large stack of plug-ins each extending the basic capability offered by the lower level plug-ins. In this scenario, every developed plug-in is considered as contribution to the community, enhancing the functionality of the existing state. The licensing of products also enhanced it’s uptake in academia and open source community. This approach is argued to be a better solution as compared to XML based approach, Net beans or IntelliJ [123]. Eclipse is emerging as dominant IDE of choice among developer community [124]. Figure indicates a generic plug-in indicating the building blocks.

Requirements to be met by Eclipse  In its current stage, Eclipse has evolved as a stack of technology projects from which required blocks are to be selected and utilized to meet the user need. Following table indicates how all the requirements are available as eclipse project. A comprehensive list of project is available on eclipse foundation website\(^8\).

6.3.9 Semantic Web Technology for Method Engineering

Method engineering is introduced in Chapter 4 as appropriate approach for orchestrating the collaborative effort[79].

\(^8\)Eclipse Project http://www.eclipse.org/projects/listofprojects.php
Semantic Web Technology  Semantic web technology is discussed earlier in this section provides support for knowledge management.

Semantics in Method Engineering  Method Engineering consists of methodology, processes, environment, artifacts and related guidance. The method should provide support for the activities of all the role players in different environment discussed in Chapter 4. Creation of content for domain specific activity requires capturing information about related role, processes, resource dependence, standard operating procedures etc. This in turn becomes exercise of capturing domain knowledge. As the method content cannot be created in isolation, it acts as a live repository continuously being updated. Other static aspects like creation of Method plug-ins in Rational Method Composer or Eclipse Communication Framework may not support this dynamic feature. As identified in the requirements for CASE tool in Section 6.1, Event driven, rule based access in virtualized environment is required.

Alterations required in Process Tooling  To establish traceability involving the identified instances, creation of architectural products, providing rule-based access to the method content provides required support of appropriately configured middleware services. The major change is required in identifying the coverage and accessing the shared content. Instead of stand-alone instance of method plug-in, a service is required that can access to other configuration providing the missing components identified. Also it is necessary to handle method content related events like, when it is created, published, shared and accessed.

6.4 SA Core Architecture

Requirements are identified in Section 6.1. Building blocks are identified in Section 6.2. Design decisions are introduced in Section 6.3. Next step is to propose architecture based on them. Based on the design decision each building block is revisited to suit the newly identified requirements.
6.4.1 Message Oriented Middleware

Messaging is identified as one of the core requirements of situation awareness middleware. Based on its role, the message oriented middleware that provides the required services is identified as one of the building block. One of the design decisions that recommend handling vast amount of identified batch job in the form of queue, the responsibility of message oriented middleware not only limits to the communication but also becomes core enabler in information processing strategy.

In realizing various communication scenarios, specific messaging patterns are established. A comprehensive list of various messaging pattern is discussed by Hohpe and Woolf[13]. This includes specific messaging patterns that addressed case specific realization of messaging system components. These patterns can be used by establishing rules for characterization of requirement and identification of appropriate messaging patterns that can support them. The MoM supporting creation and handling of messaging components is utilized by this rule driven system. Hence the resulting system acts as a Messaging Pattern Factory that manages handling of messaging component on the runtime. This includes creation of new patterns, configuration, monitoring and administration to suit the need. This in turn becomes the life cycle management of messaging pattern.

Considering the need of specific usage scenario, various messaging patterns are suggested. The patterns include various system components like message, message channel, pipes and filters, message routers, message translator and message end-point. Patterns also recommended various communication strategies that can be realized within the system by employing these building blocks. As an example, the channel can be realized with patterns. These features are handled through a pattern factory that manages the life cycle of messaging patterns.

Figure 6.6: Improvements in MOM
supported by popular platform architectures. Java platform\(^9\) and Microsoft .Net Framework\(^10\) are competing technologies. Java community process has resulted in multiple specifications addressing the messaging services. Java Message Service (JMS) is one such specification that provides creation, administration and handling of messaging patterns. JMS\[^{93}\] supports realization of various messaging related patterns may be identified with different names. Other vendors in MoM domain may support building of other patterns. It handles message channel pattern as Destination. Point to Point channel is supported as a Queue and Publish Subscribe channel is supported as Topic. Message producer and message consumer are the endpoints supported in JMS. Other vendor technologies like Microsoft MSMQ, WebSphere MQ, TIBCO, WebMethods, SeeBeyond and Vitria provide similar capabilities. Sun JMS is selected as preferred technology as it is a proven open source implementation in Java technology. Java message queue JMQ \[^{125}\] is the implementation of JMS made available. Glassfish\(^11\) release is an open source server technology that provides various implementation of Java specification.

Individual pattern creation and handling can be done using web based client or administrator tool. In application scenario, individual creation of pattern cannot help. Due to requirement of handling many patterns, it should be handled by program following the specified rules. Java Management Extension (JMX) \[^{126}\] provides MBeans that allows creation, monitoring and handling of the messaging components from program.

Existing MoM is capable in providing creation, monitoring and handling of messaging patterns with important features. Automation the management of life cycle is missing that is identified as important feature of SA middleware. Figure 6.6 represents how other building block can bring benefits to MoM in achieving the objectives.

**Rules for identification of patterns** There can be many possible patterns. Selection of appropriate pattern should be done automatically on the run time. There are specific criteria for selection. They are encoded in rules form. Hence service from semantic middleware is required.

**Management of Endpoints in VO** The pool of systems and resources that utilize them are seen as forming the virtual organization. Hence their membership to virtual organization needs to be handled as grid resources.

**Message Logs and Message Content** The automated strategy is expected to create many

---


\(^10\)http://www.microsoft.com/net/

\(^11\)Glassfish: [https://glassfish.dev.java.net/](https://glassfish.dev.java.net/)
individual information interactions in the form of identified information exchange patterns. Status is monitored and recorded during the lifetime of these patterns. Hence they are expected to generate considerable amount of data in form of logs and event traces. Data middleware services must be used to handle this generated information.

**Event detection logic and filters** Within rules for generating and messaging patterns, geographical attributes provide important basis for decision-making. The required query, filtering and rendering functionality is supported by GI middleware.

In realization of the discussed functionality, the required solution must be able to invoke operation to other middleware services. Hence a control class is designed that can handle cross middleware service invocation and handling is carried out by them. All the lists of identified task specifications are maintained as a list of scheduled batch jobs. These lists are implemented in the form of queue.

Transient resources are identified first as messaging endpoints. The members of VO may act as source or sink of the information. The determination of appropriate channels, message format and strategies for handling the message is achieved with corresponding instances of the patterns.

### 6.4.2 Semantic Middleware

Semantic middleware is important building block providing the required support for creation and handling of knowledge base and fact base. The logic of unified process is handled by the semantic services in order to determine roles, providing situation awareness in the form of required task allocation, guidance, alerts and other information determined suitable for them. Information processing logic is also captured in the form of rules. This includes determination of appropriate granularities and identifying suitable task specifications for them. Application logic that governs the processing and handling of information is also handled with semantic middleware service providing important functionality. Capturing and using knowledge in all the related domains also requires support of semantic middleware service. Local knowledge and facts captured to depict ongoing situation is also realized over the same service. All these are handled in reference to the ontology created for the same.

This functionality is offered by multiple products available in semantic web technology domain. For creation of ontology according to RDF and OWL standard is supported by Jena. Hosting of ontology is realized with Sesame service. This also support query according to SPARQL. A reasoning service is provided by Fact++, pallet and Racer products. They provide
interface for reasoning service.

Figure 6.7: Improvements in Semantic Middleware

Various resources need to be handled to meet the requirement. First the knowledge captured in the form of Ontology involves local, domain independent, domain specific and application specific concepts. They are captured and used in the form of multiple shared Ontologies continuously being upgraded. This vast pool of ontology is important resources that must be handled by the planned middleware service. The handling also include pool of providers in the form of users who enable creation of concepts, rules and provide status of instances. The pool of users who access the knowledge should also be handled by the system.

Existing semantic middleware is capable in providing creation, handling, sharing utilization of knowledge with important features. Automation in them is missing that is identified as important feature of SA middleware. Figure 6.7 represents how other building block can bring benefits to semantic middleware in achieving the objectives.

**Messaging for Assertion** Assertions made on the runtime are done by multiple users. The users must be appropriately informed about the nature, frequency and content specification of required assertion that is suitable in given configuration. This requires specific information exchange pattern. Hence, it can be said that in order to handle consistent knowledge management in collaboration environment, messaging plays an important role by realizing required communication pattern.

**Assertion sources in VO** Attributes of instances, roles assumed and the information accessed by them in the configuration is done being part of a VO member. This allows application of security framework, thereby providing features of grid into handling of
knowledge base. Intersection of semantic web and grid technology brings benefits to knowledge management process in collaborative environment.

**Instance storage and management** According to the design, multiple assertions are handled in automated manner. This results in huge instance base. The storage, management and handling of such larger instance base must be done with the help of data management services.

**Handling of spatial attributes** Semantic middleware is equipped with handling conceptual relationship among concepts. Some extensions also allows query and reasoning with temporal properties. Yet, spatial query and reasoning requires additional GI processing capabilities. These capabilities required GI middleware support. Queries for filtering, selection, operations like aggregation, event detection and snapshot generation requires the functionality of GI middleware. This is extended by appropriate integration of GI functionality in handling knowledge base.

Control classes must be defined that can provide access to services offered by the middleware building blocks. A control class created for management of assertion requires access to middleware service.

Transient entities include instances being asserted and handled by the system. The inferencing is also critical service provided by the semantic middleware. The assertion of basic instances and continuously updated information regarding their status allows identification of their membership to other classes. Various relationships they stand with other instances in form of roles and events also becomes transient entities handled by the system. In case of mapping the instances to meet the transformations various ontology mappings identified among instances, rules for transformation etc are also identified as transient resources.

### 6.4.3 Grid Middleware

Virtualization of resources and execution environment are two important function provided by the grid middleware. Open Grid Services Architecture (OGSA) is the standard. It addresses the requirements like interoperability and support for dynamic and heterogeneous environment, resource sharing across organizations, quality of service, job execution, distributed data management, Security Globus Alliance \(^{12}\) is a community dedicated in developing grid technology.

\(^{12}\)http://www.globus.org/alliance/
6.4 SA Core Architecture

Globus Toolkit\(^{13}\). Open Middleware Infrastructure Institute\(^{14}\) (OMII) UNiform Interface to COnputing REsources\(^{15}\) (UNICORE) are popular implementation providing grid services \(^{127}\).

![Figure 6.8: Improvements in Grid Middleware](image)

Within the VO formed by dynamic set of collaborating entities, resources to handled by the grid middleware includes VO members, computing resources, data resources, data sources, services and other important feature providing data, functionality or grid service. Existing Grid Middleware is capable in providing handling dynamic set of resources with various grid features. Automation in handling resources in utilization of computing capability and in other grid specific task is missing that is identified as important feature of SA middleware. Figure 6.8 represents how other building block can bring benefits to grid middleware in achieving the objectives.

**Job Specifications and monitoring communication** Various components and members in virtual organization is continuously required to communicate in order to utilize various features of grid. For exchanging job specification, event and alert notifications, monitoring and control messages, appropriate interaction patterns must be realized in the VO. MoM features in creating and handling appropriate messaging patterns bring important benefit to the grid middleware.

**Rules for generation of Task specifications** Grid middleware provide job execution environment. The jobs can be submitted using command line clients, portlets of desktop applications. In complex dynamical system, identification of individual job submission becomes difficult. Application logic captured in rules form can be utilized for automated

\(^{13}\)Globus Toolkit Web Page: [http://www.globus.org/toolkit/](http://www.globus.org/toolkit/)

\(^{14}\)http://www.omii.ac.uk/

\(^{15}\)http://www.unicore.eu/
6.4 SA Core Architecture

Storage and handling of Job Outcomes  

Storage and handling of information exchange patterns and job outcomes are required. The data middleware can allow handling of data sources and data resources that capture and handle data generated on the runtime.

Spatial reasoning in managing VO resources  

In managing VO resources, spatial attributes of the resources provide important criteria for decision making.

6.4.4 Data Middleware

Data middleware provide data handling and management capability. Products for storage and management available in the form of database management systems are useful to act as data resources in the grid environment. In order to access the shared grid resources middleware in form of libraries and products are useful. OGSA-DAI is one example of data middleware that provides this facility. It allows basic database operations, monitoring and administration in grid environment. Existing data management middleware is capable in providing distributed data management, ETL, OLAP and other important features. Automation in them is missing that is identified as important feature of SA middleware. Figure 6.9 represents how other building block can bring benefits to Data middleware in achieving the objectives.

Communication of Data Management Tasks  

Data management in distributed environment requires interaction patterns among system and users that manage them. This includes communication of data management tasks, status monitoring and their output. Some of them may be created in response to the observed events. Hence incorporation
of communication pattern brings required reactivity to the data management tasks to suit the ongoing requirements.

**Creation of Data management task specification** In complex dynamical systems where considerable amount of data is created and handled among distributed data resources, the individual attention to database management task becomes difficult. Data management task specification can be generated automatically by the system. Semantic web technology can help utilization of rules that can automate identification of required tasks.

**Virtualization of Resources** Multiple data resources made available in a configuration. Handling them individually for appropriate utilization is not possible where availability of such resources vary with time. By making it a virtualized resource in grid environment, and handling it as a grid resource brings benefits of grid middleware services to data management.

**Spatial query and processing** In processing of data, spatial attributes play important role as spatial footprint of concept may have implications. Available database management systems have varying capability in handling geographical information. Hence provision of consistent middleware services in generating and handling spatial data can bring benefits to database middleware. Also, distributed data management tasks also requires spatial attributes of data sources and data resources and therefore it can also bring benefits to distributed data management tasks.

A control class created for Snapshot generation provides implementation of Algorithm 3.6. Distributed Data management provides implementation of Algorithm 3.13. ETL Specification class provides implementation of Algorithm 3.10. Data Provenance manager class provides implementation of Algorithm 3.12.

The data middleware supports handling of various transient resources with the help of other middleware functionalities. The data sources that are sources of information are dynamic set of entity that may join the VO as sensor, human observer or other information service. The data management at lower level requires data resources, the physical instances database server that host the required information during the runtime. The availability of data resource is also dynamically changing and hence each one is treated as a transient resource. The data sets identified created handled, utilized and purged according to ETL and distributed data management policies, are also transient resources handled by the middleware. Along with the data sets, the corresponding provenance records are also treated as a transient entity.
6.4.5 Geographic Information Processing Middleware

Requirement of handling geographical information is identified since the discussion of spatial footprint in Chapter 3. This includes spatial data representation, spatial data storage, spatial query and other related tasks.

There are several commercial and open source libraries, tools and technologies available that support the required GI processing functionality at varying levels. GeoTools provide a set of libraries in handling spatial data. The Web services exposing GI functionality as per the standards by OGC is important to development of GI Infrastructure. The Deegree and Geoserver provide implementation of various OGC service for handling map, spatial features, catalogs and other aspects to access information over web service framework. From client point of view, user Friendly Desktop Internet GIS (uDig) is an eclipse based client for accessing data as a client of OGC web services. This product provides with extension point that can be utilized to develop required functionality.

Existing GI middleware is capable in providing creation, handling, sharing and utilization of geospatial information with important features. Automation in them in order to realize spatial data access patterns are missing that is identified as important feature of SA middleware. Figure 6.10 represents how other building block can bring benefits to GI middleware in achieving the objectives.

**Messaging for Spatial data generation and Access** Realization of spatial data access pattern in creating and providing spatial content in response to an event requires utilization of a rule based system for hosting required communication pattern among systems and users. Appropriate rules captured and utilized using messaging middleware can bring
benefits to GI middleware.

**Rules for spatial data generation and access** Realization of spatial data access pattern in creating and providing spatial content in response to an event requires utilization of a rule based system. Appropriate rules captured and utilized using semantic middleware can bring benefits to GI middleware.

**Virtualization of content and service** Sources of GI information, GI services and other related resources involved in providing and handling spatial content may dynamically be available to the configuration. Handing them as members of the virtual organization can bring benefits of grid features to GI middleware.

**Data sources for spatial data storage and handling** Within rules for generating and messaging patterns, geographical attributes provide important basis for decision making. The required query, filtering and rendering functionality is supported by GI middleware.

A control class that exposes functionality of libraries that allows creation and handling of GML documents. **GML Manager** is suggested that provides encoding function in creation and handling of GML documents. It utilizes functionalities of other middleware services in listening and writing to task specification queues. This along with other components in GI component in turn provides domain middleware service. Transient resources in GI environment are handled by the configuration. Representation of situation at specific granularity is created, processed and published by the GI middleware. They can be considered as coverage providing thematic data derived from monitoring of real life features prevailing in UoD. In order to create such themes of representation at varying scale, appropriate human or sensor observations are required. The tasks of observations, messages that hold the observed values, communication patterns that allow realization of information flow and templates are all the transient entities that are handled by the system to create the desired geographical representations. Apart from these themes many Architectural Products are also rendered as map, hence they are also one of the important transient entities handled by the GI environment in realizing Situation Awareness services.

**6.4.6 Runtime Middleware**

The proposed system is designed to provide domain middleware service targeted at situation awareness. This is realized with reactive nature. Various information processing tasks are triggered by the occurrence and detection of event. The availability of dynamic set of participating resources is continuously checked and handled appropriately. In this scenario, the runtime services provide basic features required for configuration of underlying services and components.
Configuration of the instances of basic building blocks providing core services is primary requirement of the runtime middleware. Any state change may affect the overall services, hence continuous monitoring needs to be carried out. As many types of resources are configured within the configuration, specific roles can be identified and made responsible for those resources in the configuration. Hence, a dashboard view of the system providing configured resources is also required. The administration of various configuration services is also required functionality. For appropriate control of the runtime, the functions like starting and stopping of service is required. To support maintenance, include newly available updates and other administration scenarios should also be supported.

Various control classes can be identified that are responsible for providing middleware services. **SA Configuration Manager** is a class designed to handle configuration of core services. Link to the service is established by this class. **SA monitor** is the control class that provides monitoring features of the configured resources. The basic monitoring mechanism of lower level service is utilized to provide overall monitoring of resources in domain middleware. **SA Process Manager** is responsible for exposing the process content of Situation Awareness Unified Process. This is responsible for creation, handling and sharing of various process artifacts, architectural products and related content. **SA Runtime** provide runtime features of the domain middleware.

Deployment view of a SA configuration is depicted in Figure 6.11. As indicated, instances of various building blocks are required to fulfill the configuration requirement. MoM service instance is a link to messaging service that enables creation and monitoring of messaging patterns. Monitoring facility exposed by the MoM is utilized by the runtime. The link to service hosting knowledge base and providing query, inference and instance support is required as one of the core service. Access to GI services can be exposed with connection to many possible services providing query, data access and other GI processing functionality. Hence, there can be links to multiple GI services required to meet the needs. Link to Grid Middleware determines the membership to a virtual organization. The link provides access to various grid services and resources. Multiple links to data management services and instance may also be handled by the system.

The runtime view includes handling of various transient entities. One class of transient entities supported by the runtime is configured entities. These entities are the resources like computational resources, storage devices, sensors, services etc. Monitoring and handling of such configured entities providing infrastructure for domain middleware services are handled
6.4 SA Core Architecture

Figure 6.11: Deployment view of the SA Runtime

separately by the middleware. Another class of transient resources are the programming or logical entities that is handled by the configuration. They are handled according to the rules defined in the KB. These runtime entities are in the form of identified tasks, job, role, events or other abstract entities. As they provide the information processing functionality; appropriate monitoring and handling is supported by the runtime.

6.4.7 Proposed Architecture

Based on the core components and interactions discussed among them, an eclipse platform based architecture is proposed. Figure 6.12 represents the plug-in architecture revealing important components. The set of plug-ins are classified based on the functionality provided by them. The logical collection of core set of plug-ins and related resources are identified as SACore. SACore includes Core Plug-ins, Runtime Plug-ins, Utility plug-ins and resource pool.

Core plug-ins are set of plug-ins that realizes the control class that provided extended functionality of middleware as discussed in the earlier part of this section. Assertion Manager, Information Need Manager, MOM Pattern Manager, Event detector, ETL Manager, Grid Manager etc. are the core plug-ins that provide core functionality of the SA Middleware. Their functionalities are realized as higher level domain services that are built over middleware services provided by the building blocks in lower level of abstraction.

Utility plug-ins are set of plug-ins that expose services of middleware building blocks such that they can be consumed by the core plug-ins. The utility plug-ins includes: MoM plug-in, KM plug-in, Data Plug-in, VO Plug-in and GI Plug-in.

Runtime Plug-ins as the name suggests, provide functionality for configuration and execu-
tion management. These plug-ins provide support for configuring, monitoring and handling of required computing resource. They also provide point of access that exposes the core services to the various clients. The situation awareness configuration is discussed throughout the discussion is made accessible using these runtime plug-in components.

Figure 6.12: SA core Architectural Components

Apart from the static content and the specific functionality rendered by the plug-ins in the SACore, there are special features represented in the system. The utility plug-ins are not represented as serving the requests issued by the control plug-ins in the above layers, they also provide, Situation Awareness configuration specific content. For example, the MoM utility plug-in exposes various functionalities provided by JMS, JMX and JMQ libraries. It also represents core queues required to handle automated tasks identified on the run time. Apart from this
configuration aspect, the SA Middleware manages the life cycle of messaging patterns. These managed objects are depicted to be part of MoM. Also it is possible that multiple MoM services are required to be bridged to meet the multiple configuration requirements. In the resource pool various message brokers and other messaging system components are depicted.

Similarly, the KM service instance represents the Situation Awareness Core Ontology that consists of KB. All the facts and assertions created in references to this KB. It also contains the rules for various application specific information-processing tasks. Inferred instances in form of identified events, actions and generated representations are indicated as transient resources handled by the service.

Data service instance, GI service instance and VO services instance are similarly represented as handling appropriate artifacts and transient resources associated with them. Required and available resource pools that support these services are also represented to establish the responsibility and capability of the system to meet the needs. The content hosted and served by the runtime plug-ins include Situation Awareness Unified Process and various artifacts, work products and components exposed in the enterprise continuum.

### 6.5 System Views

Architecture discussed earlier can be utilized to meet the requirement. The system should be able to demonstrate its capability in meeting the user needs of complex dynamical system. There are the abstract representations created for demonstrating system capabilities. Once they are configured and created, deployed according to the process.

The architectural products provide view of the individual component features. Whereas, a separate level of viewpoint is required to evaluate and monitor the overall system. For the visualization of salient features of the proposal, various views are proposed. They demonstrate specific capabilities show how system design is meeting the desired requirement. The views are identified as follows:

#### 6.5.1 Event View

Primary focus of the event view is to demonstrate all possible events that are handled by the system. According to the definition of event proposed in Chapter 3, event is attached to entity or process, has detection mechanism in reference of a configuration and appropriate role that is responsible for suitable response. This leads to involvement of various environments discussed in unified process.
Hence, it can be said that event view is focused on depicting event space along with related entities. Figure 6.13 provides a part of event view mainly covering the configuration level events. In similar manner, events can be depicted with related concepts across all environments.

Figure 6.13: Event View of the Situation Awareness System

The event view indicates the interactions among related components that are triggered due to events. Prerequisites for such interactions are specific for handling events. Event sources are identified a priori in the knowledge base and their occurrence is verified with the help of event profiles. Based on this knowledge, available instances of event sources must be configured and monitored appropriately. The profile-matching task needs to be carried out at defined interval. In case of detected event, the identification of action, selection of user and notification should be created. This view amounts to be a considerably comprehensive view of situation awareness middleware capability that is responsible for desired reactivity. Realization of such view demonstrates that the designed system is an example of Distributed Event Based System. Also, it establishes the event space - a set of all possible events that can be handled by the system.

Capturing Events Any state change in an entity can be considered as event. The goal is to identify those changes that are relevant to an actor or a process, and to notify appropriately so
that necessary action can be taken. Here *entity* can be a job, a resource, a person, equipment or any other real life object. The change of state is very specific to the type of entity. For example a job can be in *Started, Suspended, Failed, Aborted* or *Finished* state. These state changes are very different from the changes that can be experienced by other entities. The types of events (state change) that can take place in a given environment are identified are shown in Table 6.12.

### Services as Source of Event

Services are the smallest important functional units in the proposed SOA based system. Change in their state is an important source of events. In its simplest form Web Services can be developed based on publish-find-bind paradigm. By following the standard-based approach, the service description in form of Web Service Description Language (WSDL), the discovery supported by Universal Discovery Description and Integration (UDDI) specification and subsequent interaction achieved using Simple Object Access Protocol (SOAP).

Beyond this minimalist approach, appropriate implementation with other specifications in *Web Service Specification Stack*, more specific functionality can also be incorporated. Notification design pattern can be achieved in web service by realizing the WS-Notification Specification[128]. WS Notification is a set of three specifications namely: WS BaseNotification, WS Brokered-Notification and WS Topic. WS-Topic deals with subjects or items that can be of interest for notification. A Web Service can publish a set of topics that a client can choose to subscribe. In case of a change in subscribed topic, the subscriber is sent notification of the event. The specification also supports the hierarchical or tree based structure of topics. Subscriber can directly subscribe to desired topics or can use the broker service to manage their subscription. In case of brokered notification, publishers register the published topics to the broker and similarly subscribers manage their subscription to published topics through a broker. Regarding the

#### Table 6.12: Event Types

<table>
<thead>
<tr>
<th>Entity</th>
<th>Domain</th>
<th>Type of Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization</td>
<td>Real-world</td>
<td>Adopting response</td>
</tr>
<tr>
<td>Actors</td>
<td>Real-world</td>
<td>Taking Actions</td>
</tr>
<tr>
<td>Concept</td>
<td>Ontology</td>
<td>Inferred membership</td>
</tr>
<tr>
<td>Situation</td>
<td>Ontology</td>
<td>Inferred Info. Need</td>
</tr>
<tr>
<td>Resource</td>
<td>Grid</td>
<td>Joining/leaving VO</td>
</tr>
<tr>
<td>Job</td>
<td>Grid</td>
<td>Execution State Change</td>
</tr>
<tr>
<td>Res. Prop</td>
<td>Web Service</td>
<td>Change in Res. Prop</td>
</tr>
<tr>
<td>Subscription</td>
<td>MoM</td>
<td>Change in subscription</td>
</tr>
<tr>
<td>Event</td>
<td>MoM</td>
<td>Detection of Event</td>
</tr>
</tbody>
</table>
possible topics in Web Services, any state change in defined resource property can be published.

**Grid as Source of Events**  The Open Grid Services Architecture (OGSA) Specification offers a framework for building standards-based grid services [128]. The security services provided by Grid Security Infrastructure (GSI) that is responsible for the membership to the VO, the monitoring and discovery service (MDS)[129], the job scheduling and execution monitoring services[130] are some important services that the middleware communicates with. The status of submitted job, status of grid resource, or the membership to the VO are some sources of events that other services may track for change. Hence, mechanism to detect such grid service specific events[131] is of critical importance to the communication middleware. For facilitating the required monitoring of grid resources [132], access to these services can be provided to the end-user using appropriate portlet technology.

**Data Sources as Source of Event**  The data sources like a sensor, a network of sensors or human observers are considered as grid resources. They are uniquely identified across EOC VO, and can create and consume various control and information messages from the middleware. The status of the data source is also a relevant source of events as it allows tracking the availability of source in a dynamic environment.

**Inference as Source of Event**  The events discussed till now, forms a set of event that can be monitored, verified and accessed programmatically. These state-changes are explicit and easy to capture. There can be another set of events that are not identified explicitly but can be inferred based on certain rules. The rules that govern inference of implicit events can be realized by hosting an inference service in the Grid. The inference engine is referred to with new assertion for detection of any implicit events. The reasoner is capable for calculating the membership of given fact to specific classes that are defined as event. In summary, event view of the situation awareness system is helpful in bringing out event driven nature of the system. It also demonstrates event space of the system.

### 6.5.2 Middleware View

The proposed system is realized as domain specific middleware services. The domain here is the situation awareness. The requirement of situation awareness capability is established in earlier chapters. The middleware view represents how requirements are met. The basic elements in the view are therefore the features of a situation awareness system. As these features can be realized with existing tools and technology support, the same are included as basic building blocks. As
the system is proposed as a middleware, it is exposing services to clients and applications in the upper layer.

The entities in the middleware view interact with each other in form of procedure calls and responses. In the lower layer, that consists of distribution middleware that include SOAP based message exchanges. The domain specific middleware services at higher level utilize such lower level services in response to events detected and rules identified for the same. In this manner it solves the problem of Wild card capabilities of the lower level middleware components.

Prerequisites for the realization of middleware view are the configuration of the various building blocks. This configuration is guided by the rules defined for achieving the domain specific functionalities. These functionalities include realization of information processing models discussed in Chapter 3.

![Figure 6.14: Middleware View of the Situation Awareness System](image)

With this view, the situation awareness capability of the system is demonstrated. The functionalities like life cycle management of transient resources, event driven assertions, automated ETL and job specification generation, automated GI theme generation are the capabilities that are built over lower level middleware services. This is realized using various queries, generation, manipulation and representation of information. Figure 6.14 depicts the layers of middleware. MoM, Semantic Middleware, Grid Middleware, Data Middleware and GI Middleware are depicted as building blocks. Various situation awareness middleware capabilities enumerated above are realized by appropriate interactions among these blocks. These facilities can directly
be consumed by clients with the help of GUI. Alternatively, these functionalities can be exposed to client applications. Dynamic web content realized in the form of Widgets, Mash-ups, portals or feeds could act as clients. Similarly, in accessing large amount of data, the business intelligence and reporting applications can be utilized. Monitoring, dissemination and processing applications can also be configured to utilize information processed by the middleware.

This capabilities are fulfilling the needs identified for situation awareness of individuals and organizations as defined in Chapter 2. According to the definition, individual requires information at specific granularities. Apart from basic monitoring data, individuals may require task specific actions, event notifications, alerts, guidance and other related information. Middleware services in proposed view depicts how these requirements are fulfilled with the help of middleware at different level of abstractions and exposed to client applications.

### 6.5.3 Runtime View

Views discussed so far provided representation from the internal part of the system. Event View depicted event space handled by the system, whereas the middleware view depicted how the components are utilized in realization of various middleware services. Yet, these views do not reveal how configured system components are serving the purpose. The runtime view is proposed as a solution to this problem.

![Figure 6.15: Runtime View of the Situation Awareness System](image)
As runtime view is being proposed to reveal the configuration of various entities at runtime; it consists of the users, their interfaces to the system, system components, and various transient entities that are managed by the system. Actions, events and reaction to the identified events by users and system provide the basis of interactions. Prerequisite for this view is the system configured appropriately. This includes configuration of building blocks, various resource pools and various role sets.

This view demonstrated capability of handling life cycle of transient entities. Various queues realized to perform scheduled jobs according to rules. Continuous gap analysis provided to the users. Alerts provided to the users. Access to representations and guidance provided to the users. These capabilities fulfilled the need of systems in meeting situation awareness. The management of transient resources fulfilled the need of automation in services. The rules handling resource pool in the runtime addressed the need of handling available wild card capabilities of the resources.

6.5.4 Configuration View

In runtime view, many transient resources are created and handled by the system. Apart from this, work products from the users are collected, architectural products and various components are stored in enterprise continuum. Various representations are also created in the runtime. Many of these are potentially useful in other scenarios. Hence, it is necessary to establish reusability of the discussed entities. The configuration view is proposed as a solution.

Determination of reusable components followed by accessing the required one and sharing the created resources are next logical step. This is achieved by appropriate configuration. Hence, these entities are the primary focus of the view. As discussed earlier, such entities include transient entities, representations, events, alerts, notification, architecture products, work products, guidance and other items generated during the runtime or created in the configuration. This also includes knowledge base consisting of rules, mapping and concept definitions.

Based on reusability and relevance these products can be separately identified in to four classes. Figure 6.16 represents quadruple of generated artifacts. They are classified on the basis of spatial and temporal relevance. The generated artifacts are valid either for long term or short term. Similarly, in spatial dimension, the generated artifact is applicable locally or globally. Based on such classification, it is easy to determine how configuration will be required to share and reuse the available and required artifacts respectively.

These work-products are outcome of various users active in distinct work environments. As
a prerequisite, all such actors must be connected to the configuration. They must be sharing the work-product outcomes that are created as per the guidance provided by the configuration. These outcomes are created for specific purpose that can be attributed to some rule and the rule also can be traced to occurrence of event. There can be multiple such events that trigger various rules resulting in numerous work products. Hence appropriate Meta data clearly identifying the purpose and nature of artifact is required. The appropriate strategy for determining the suitability of reuse in spatial and temporal dimensions is required to be established.

By fulfilling these prerequisites, it is possible to define configuration of a system. Figure 6.17 depicts the type of configuration required to access or share four type of artifacts depicted in Figure 6.16. The artifact that are relevant for short period of time and that are valid locally requires single configuration. The enterprise continuum is not required to share and import any other artifacts from external continuum. A scenario in which local but long-term configuration is required, enterprise continuum created (and currently archived) for short term can be utilized to cover the longer period. Hence multiple instance of enterprise continuum for the same location is configured to meet the need. In case an enterprise continuum contains globally relevant components, and may require additional relevant components that are created and hosted by other enterprise continuum, then required configuration as depicted in quadrants named global short and global long term.

This view demonstrated configuration capability of the system. This indicated sharing of generated artifacts and utilization of globally archived and shared artifacts that can be useful. This demonstrates the capability of enterprise continuum. Also, it provides mechanism of deter-
6.5 System Views

6.5.5 Role View

Role sets are introduced in Chapter 4. This included various actors performing some actions and creating work products. To aid the action, situation awareness is provided in the form of guidance, alerts, notifications and other relevant information. Architectural products are important means of providing situation awareness as they provide traceability and gap analysis. How various role sets are utilizing the situation awareness is important. This is captured in Role View. Basic elements in role views are role sets, artifacts produced by them and the architectural products utilized. Interaction in role view is depicted in the form of user-system communication. System provides consistent representation of architectural products, establishing traceability and gap analysis. The users on the other hand, provide task related information about status of the generation of work products. Prerequisite to the role view is configuration in which all role sets engaged in producing work products are connected to the Situation Awareness configuration. They access the architectural products identified suitable for the role. Also they provide information about the status.

This view demonstrated capability of SA Configuration in linking role sets active in various
Figure 6.18: Role View of the Situation Awareness System
work environments to be connected to single unified configuration. For example organizational process, configuration and end user role sets are in active in different environment yet, they are depicted to be connected uniformly in SA configuration. Even though various users are connected to the same configuration, they have separate architecture products. This demonstrates the Separation of Concern. This is allowing the user to access and use only required information without manually filtering off all the information handled by the system. This fulfills the requirement of establishing cross environment traceability. For example, the standard created by standardization organization in organizational role set can be traced to the product that complies with the same. Hence, though in different environment, the organizational actor can trace the use and check the performance and related detail of the work product outcome.

6.5.6 Service View

Services are important building block of the system. Many domain-specific middleware functionalities are exposed as service or consume service. Service, Service component, execution environment and related role are the basic elements of the service view. All the stakeholders related to the given service may not interact directly. Some of them are having implied links among them while others are directly connected. Other can interact by having work product configured and monitoring the performance or just by fact of being utilized.

Prerequisite for service view is the prior knowledge of how service is composed. How various building blocks of the service are available as a outcome of some activity of a role.

This view demonstrates the capability of SA architecture for being able to dissect a service to all possible unit contributions as outcome of user activities. It shows how a single service can be monitored and characterized beyond WS * protocol stack to reveal the complexity. By providing links of each aspect of a web service to associated user role sets, the view established the capability of providing traceability and separation of concern. With establishment of web service characteristic various component level, the basis for appropriate service governance is also created.

By considering an instance of a Web Service instance as an SA entity, various attributes can be established along with the impact footprint as defined in Chapter 3.5. A footprint of a web service, indicating its service covering a specific geographical area, for specific time providing a specified functionality or information as defined in SA Configuration. In case of service unavailability, the exact gap analysis and traceability can identify precise action requirement from appropriate role sets, in order to fulfill the service gap. Also, the contributing role sets can identify how the unplanned reuse of their contribution is taking place in realizing unique
6.6 Summary

This chapter presented views on realization of situation awareness capability in distributed computing environment for the requirements identified from initial chapters of the thesis. The proposal is made as extensible system that is built over existing components. These components have matured with research and development effort of vast community and compliance with standards. The extensibility of the proposal ensured that the proposed system can be carried out as community contributed extensible plug-in based project. In its primitive version, the system is currently identified with basic minimum functionality of situation awareness domain. Collaborating contributors can extend the functionality to meet the newly identified needs. In summary, the chapter discussed middleware, patterns and framework as described by Schmidt and Buschmann [91] in reference to situation awareness capability. The unique features and capabilities of the proposal are depicted with the help of various views.

Figure 6.19: Service View of the Situation Awareness System