

## Model-Driven SOA for Sensor Networks

John Ibbotson<sup>a</sup>, Christopher Gibson<sup>a</sup>, Sahin Geyik<sup>b</sup>, Boleslaw K. Szymanski<sup>b</sup>, David Mott<sup>a</sup>,  
David Braines<sup>a</sup>, Tom Klapiscak<sup>a</sup>, Flavio Bergamaschi<sup>a</sup>

<sup>a</sup>IBM United Kingdom Ltd., Hursley Park, Winchester, Hampshire, SO21 2JN, UK;

<sup>b</sup>Rensselaer Polytechnic Institute, 110 Eighth Street, Troy, NY USA 12180

### ABSTRACT

Our previous work has explored the application of enterprise middleware techniques at the edge of the network to address the challenges of delivering complex sensor network solutions over heterogeneous communications infrastructures. In this paper, we develop this approach further into a practicable, semantically rich, model-based design and analysis approach that considers the sensor network and its contained services as a service-oriented architecture. The proposed model enables a systematic approach to service composition, analysis (using domain-specific techniques), and deployment. It also enables cross intelligence domain integration to simplify intelligence gathering, allowing users to express queries in structured natural language (Controlled English).

**Keywords:** SOA, Model-Driven, Middleware, Sensor Network, Service Composition, Controlled Natural Language

### 1. INTRODUCTION

We observe that IT architectural design principles and techniques that have matured within enterprise systems are now migrating from the core to the edge of the network, and are being applied to systems implemented on dynamic, unreliable network infrastructures that are primarily used for sensors. In this paper, we identify four motivating and converging trends that lead to a new approach to the design and implementation of information systems that include data originating from sensor networks.

Firstly, sensor networks are commonly implemented as stovepipe systems with limited or no information sharing capabilities. In previous work [1] we have described the ITA Information Fabric, a middleware architecture that spans the network from the data center to deployed sensors and mobile personnel. It tracks the sensors, nodes, and users of the sensor network facilitating universal access to sensor data from any point, and maximizing its availability and utility to applications and users. The Fabric is an extensible platform. Its plug-in architecture allows new functions (such as filters, transformations, policies, security, and event detection algorithms) to be deployed directly into the sensor network and selectively applied to sensor messages en route to the user. It is essential that data originating from sensing systems be available to a multiplicity of users and fused with other information sources to provide robust situational awareness that can be easily, yet securely, shared within and across organizational boundaries. This might include governmental, non-governmental, military, and commercial organizations. Consider, for example, the cross-organizational operations carried out in the context of a humanitarian relief effort.

Secondly, the service-oriented architecture (SOA) is a popular design methodology that is suitable for deployment out to the edge of the network (i.e. on sensor networks). SOAs consist of a set of architectural principles that have been widely adopted in the enterprise IT domain through their implementation using the open standards of Web services. Within the enterprise, highly managed, scalable and resilient network and IT infrastructures provide the platform for implementing Web service based SOAs. In resource constrained sensor networks, however, alternative implementations must be considered whilst still retaining the fundamental architectural principles of an SOA. The tradeoffs that must be adopted in these circumstances have been discussed in [2,3] and have been shown to enable effective service deployment at the edge of networks.

Thirdly, model-driven architectures (MDA) provide useful tools and techniques for the analysis, design, and deployment of services, including physical assets, within a sensor network. An MDA is a software design approach for the development of IT systems. It provides a set of guidelines for the structuring of specifications that are expressed as

models. These models fulfill different roles in the design process from an initial, abstract view of specifications through to a model representing the software system deployed on a target IT infrastructure. Standards and tools underpin MDA so designers can specify models in formats that can be easily shared and transformed [4].

Finally, information from sensor networks forms only part of a user’s situational awareness. In a military context, for example, a commander also requires other sources of intelligence such as HUMINT<sup>1</sup> and SIGINT<sup>2</sup> to make effective decisions. In order to allow information to be joined across different intelligence domains, a common representation is required. Our experience leads us to believe that a form of controlled natural language (that is both human readable and machine understandable) meets this requirement.

The objective of this paper is to describe how these four trends can be brought together using a model-driven approach to design and deploy services in a sensor network. Section 2 describes the use of service modeling in the design and deployment of sensor networks. Section 3 describes controlled natural languages and their use in model definition and deployment. Finally in Section 4 we describe automated dynamic service composition that leverages service models and mitigates the management overhead of SOA on ad hoc sensor networks.

## 2. SERVICE MODELING

Services on a sensor network do not follow the conventional view of a service requestor invoking a service provider that responds to the invocation. This request/response service model is widely used in Web service oriented SOA applications but is not appropriate for the stream oriented processing services found in an event based messaging environment such a sensor network. New middleware based approaches to the development of sensor network solutions (such as the ITA Information Fabric) provide messaging infrastructures that are comparable to those used in mainstream SOA architectures, interconnecting the sensor network’s assets and users. The service composition model that is required to *efficiently* deliver composite services built from these individual functions is very different to the process choreography that is commonly used in enterprise systems. In constrained stream oriented environments such as sensor networks it is necessary to revert to first principles and discuss how a user could describe a set of services that provides a particular processing function.

For modeling services on the sensor network, we use the UML activity diagram as a starting point. UML activities match the semantics of sensors and services in the network, with them being synonymous with services having zero or more inputs and zero or more outputs. This is illustrated in Fig 1, which shows the elements of UML model describing a sensor network deployed for military base security and protection.

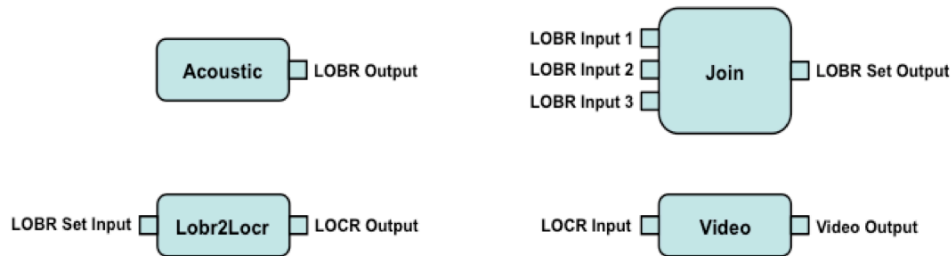


Fig 1: Palette of sensors and services

In this model, we have a palette of four service definitions available:

1. An *Acoustic* service which can provide a line of bearing (LOBR) to the origin of an acoustic event.
2. A *Join* service which groups LOBRs that occur within a given time window into a set.

<sup>1</sup> Human intelligence.

<sup>2</sup> Signals intelligence.

3. A location report service (*Lobr2Locr*) which generates a location report (LOCR) by triangulating a set of LOBRs.
4. A *Video* service that positions a camera towards a location given in a LOCR and generates and image.

From this palette, we can construct a base protection solution consisting of three instances of an Acoustic service (A1, A2, A3), a Join service (J1), a *Lobr2Locr* service (LL1) and a Video service (Cam1). The services are interconnected by drawing connections (C1..C5). At this stage, the model is abstract since the Acoustic and Video services are generic representations of assets that can generate or respond to messages of a particular type. When deployed in an active sensor network, these services will be mapped to real assets that can provide the required functions. In the case of the Join and *Lobr2Locr* services, these will be deployed as software services on processing nodes capable of providing the IT resources needed to support them. The underlying middleware (such as the ITA Information Fabric) will map the connections needed to transfer messages between the different components.

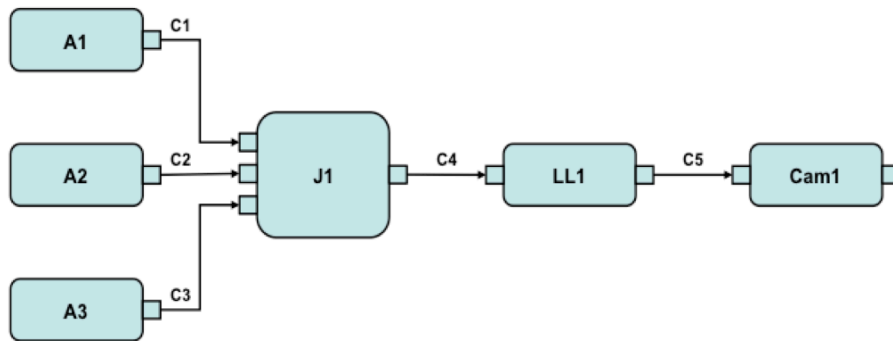


Fig 2: A base protection solution composed from sensors and services

The example illustrates several elements of a core model: activities, inputs, outputs and connections. UML allows each of these design model components to be annotated with additional information and we propose the exclusive use of semantic annotations to enrich the model, made according to a flexible and extensible ontology. Model annotations may reflect both abstract and concrete properties of a design. This combination (i.e. UML model plus semantic annotations) provides a flexible basis for both transformation and analysis of the model. A variety of models can be targeted including:

1. Formal models: for evaluating and verifying quantitative and qualitative properties of the UML model prior to deployment onto an active network. For example, transformation of the model into the Performance Evaluation Process Algebra (PEPA) used for modeling distributed systems.
2. Code generation models: for generating source code for new or incomplete components.
3. Deployment models: for mapping onto physical sensor networks. For example, a descriptor describing the deployment of a model onto the Information Fabric.

We expect transformations into formal models to be used primarily as part of an initial design process prior to deployment in the sensor network. Results from formal model analysis techniques may then be used to enhance or modify the original design model.

The choice of semantic model annotations gives a flexible technique to easily integrate analysis results back into the original UML model as feedback to the designer/developer. We describe this as *round-tripping* (Fig 3). Semantic annotations can also be used to capture physical properties of a deployed sensor network. This supports the second, concrete, round-tripping route shown in Fig 3 between the design model and the services deployed on an active sensor network. The Information Fabric provides instrumentation on the sensor network message bus allowing real-time metrics to be gathered and a profile of the performance of deployed services to be generated. This profile can be used to provide further annotations within the design model to allow its refinement, making it more suitable for the deployment

environment. We expect the UML design model to undergo multiple round-trips between the design and deployed states during its lifecycle.

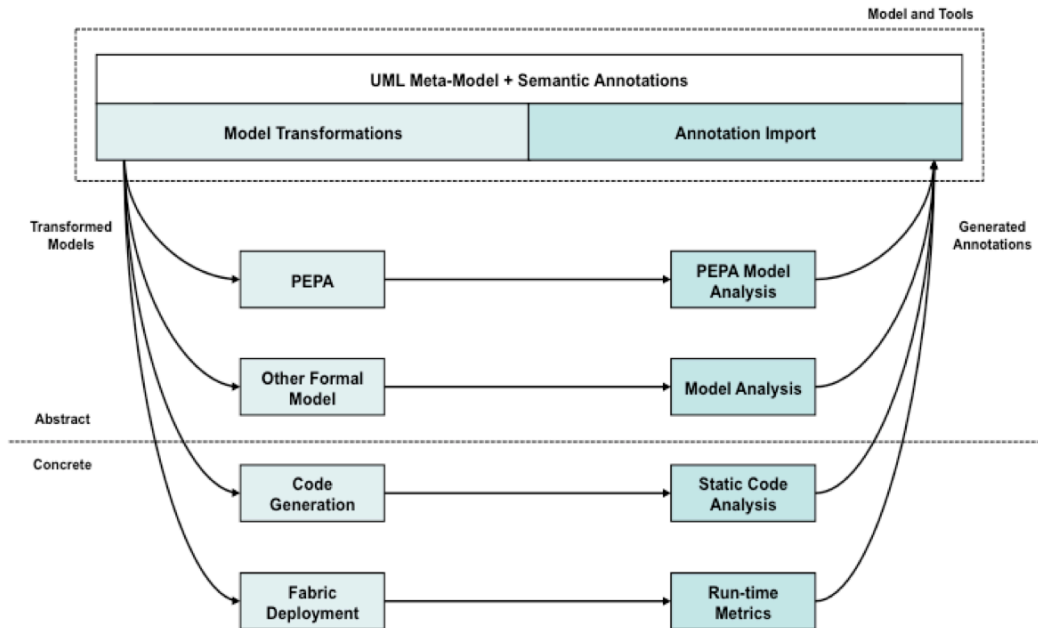


Fig 3: Round-tripping from/to the core design model

In enterprise IT development, UML is supported by a complex set of standards, design processes and tools. Our research has focused on an alternative approach to model definition using controlled natural languages. This approach is motivated by the fourth trend identified in this paper’s introduction: the need to join information from many different domains in order to provide coherent situational awareness to the user of a sensor network. We now describe in more detail the use of controlled natural languages and in particular ITA Controlled English.

### 3. ITA CONTROLLED ENGLISH

ITA Controlled English (CE) [5] is a type of Controlled Natural Language (CNL), designed to be readable by an English speaker whilst representing information in a structured and unambiguous form. It is structured by following a simple but fully defined syntax that may be parsed by a computer system. It is unambiguous by using only words that are defined as part of a conceptual model. CE therefore serves as a language that is simultaneously understandable by both a human and a computer system, thus facilitating the communication between them. The CNL used to define the core service model has been developed by other ITA research, and is based upon a draft proposal for Controlled English defined by John Sowa [6] as a human-friendly version of Common Logic (CL). During the development and use of CE within the ITA, and in several associated projects, some additions and modifications were made to the extent that we now use a different dialect, herein called ITA Controlled English. All uses in this paper of CE are references to ITA CE.

Experience in using CE within the ITA suggests that it is of significant benefit in the following ways:

- It permits the use of a single language by both human and machine, encouraging the development of novel hybrid ways of reasoning across different information domains.

- It facilitates the development and understanding of logical conceptual models containing both structures and logical relationships.
- It facilitates the viewing, construction and understanding of rationale<sup>3</sup>.
- Perhaps surprisingly, it helps users to construct models, rules and to understand the rationale for reasoning, despite the syntax of CE still being at a basic level and in need of further extension.

In this paper, we present a working description of CE rather than a complete, formal representation<sup>4</sup>. We now briefly review CE concepts and propositions and make an observation about the impact of monotonicity.

### 3.1 CE concepts

A conceptual model, or ontology, is defined in CE using *conceptualise* sentences. These define concepts, properties and relations using CE linguistic terms and phrases. Concepts that are sub-classes of other concepts may be defined. For example in the core service model, all classes are sub-classed from the entity class. This is so that properties common to all classes may be inherited from entity.

All entities have an annotation property that can be used to annotate any concept using a name, content, type triple. Example entity, annotation and service concepts are:

*conceptualise an ~ entity ~ E that  
has the annotation A as ~ annotation ~.*

*conceptualise an ~ annotation ~ A that  
has the value NA as ~ name ~ and  
has the value VA as ~ contents ~ and  
has the value TY as ~ annotation type ~.*

*conceptualise a ~ service ~ S that is an entity.*

In the sensor domain spatial information is important; concepts describing spatial classes are:

*conceptualise a ~ spatial thing ~ S that  
has the value LAT as ~ latitude ~ and  
has the value LON as ~ longitude ~ and  
~ is colocated with ~ the spatial thing S.*

*conceptualise a ~ geolocation ~ G that is a spatial thing and  
has the value AL as ~ altitude ~ and  
has the value BE as ~ bearing ~ and  
has the value VE as ~ velocity ~.*

### 3.2 CE propositions

CE propositions are statements that have the potential to be true or false. Facts are propositions that are specified to be true in a given context. These may be statements of entity existence:

*there is a service named Acoustic.*

---

<sup>3</sup> In this context, rationale is defined as the recorded result of applying reasoning steps to generate inferences from premises or assumptions, where a premise is a known (or inferred) statement, and an assumption is a statement that is not known to be true, but is presumed to be so by a human or software agent.

<sup>4</sup> This working description is subject to change as the research programme progresses.

or of possession:

*the spatial thing Hursley  
has 51.026 as latitude and  
has -1.398 as longitude.*

*the service Acoustic has the geolocation Hursley as location.*

The class properties may be other classes (e.g. Hursley) or constants (e.g. 51.026, -1.398). Current examples of instances of the service core model only contain simple statements of facts as illustrated here. This will change as further research work results in richer model descriptions. CE also has the ability to represent *propositional attitudes* such as the assumption or belief or denial of propositions (although in this document we do not use such structures). Note that modeling with CE still uses the tried and tested notions of entities, properties and relations. However the difference when using CE is that the definitions are all in a version of English. Furthermore, logical relationships are defined explicitly as CE rules, also in a version of English.

### 3.3 Monotonicity

CE sentences are monotonic. This means that properties may be added to a CE proposition but never removed or modified. When using a programming language, programmers often instantiate a class or composite variable then, over time, modify its internal variables using appropriate methods. Within the Information Fabric, this situation will most commonly occur when describing messages generated at a service output or accepted at a service input. When this is being modeled, each message *must* be modeled as a new instance with a new name; typically naming conventions may include a timestamp for uniqueness.

## 4. SERVICE MODELING WITH CONTROLLED ENGLISH

In the previous section, we introduced the use of Controlled English concepts to model services within a sensor network. In this section we expand the description to illustrate CE's use in modeling the core service concepts. For brevity, we do not go into great detail on subjects such as message schemas; these are an ongoing research topic within the ITA.

Services communicate by passing messages; messages have two properties, *metadata* and *content*, that define the schema of a message that flows between services. Messages are generated from service outputs and consumed by service inputs. Rules can then validate that the content type of a message generated by a service output is the same as the type required by a service input to which it is connected.

*conceptualise a ~ metadata ~ M.*

*conceptualise a ~ content ~ C.*

*conceptualise a ~ message ~ M that  
has the metadata M as ~ metadata ~ and  
has the content C as ~ content ~.*

A *service* concept has an *icon* for graphical representation.

*conceptualise a ~ service ~ S that is an entity and  
has the value URL as ~ icon ~.*

*Service definitions* describe a service abstraction. For example a service definition could describe an acoustic sensing service and include the generic functions and properties the service would provide. A service definition has a set of input and output *terminal definitions*, which have a message property, that defines the schema of the message generated or consumed by the terminal. These are instantiated as *terminal instances*, as part of a *service instance*.

*conceptualise a ~ terminal ~ T that is an entity.*

*conceptualise a ~ terminal definition ~ T that is a terminal and  
has the message M as ~ message ~.*

*conceptualise a ~ service definition ~ SD that is a service that  
has the terminal definition TDI as ~ input ~ and  
has the terminal definition TDO as ~ output ~.*

A service instance is an instance of a service definition that can appear on a design template. For example, a base protection template may include three instances of an acoustic sensing service (A1, A2, A3 in the earlier example). Each service instance has a service definition; there is a one-to-many relationship between a service definition and a service instance, and a one-to-one relationship between a service instance and a service definition. Cardinality relationships cannot as yet be specified within CE, but rules to enforce these relationships can be written.

Service instances have terminal instances which link to their definition. The reason we separate terminal definitions and instances is that the terminal definition provides the base definition of the terminal properties (such as the message schema it generates or consumes); the terminal instance can be used as an anchor for other properties that are appropriate for that instance. For example, some instances of a service may have different policies attached to their outputs.

*conceptualise a ~ terminal instance ~ TI that is a terminal and  
has the terminal definition TD as ~ definition ~.*

*conceptualise a ~ service instance ~ SI that is a service and  
has the service definition SD as ~ definition ~ and  
has the terminal instance TII as ~ input ~ and  
has the terminal instance TIO as ~ output ~.*

*Connections* are used to interconnect service outputs and inputs; they have *start* and *end* properties. The start and end properties are *terminals*.

*conceptualise a ~ connection ~ C that is an entity and  
has the terminal ST as ~ start ~ and  
has the terminal ET as ~ end ~.*

#### **4.1 Composite services**

A *composite service* is itself a service that is made up from a set of interconnected (wired) *component services*. The core model allows a set of instantiated services and connections to be collected into a composite service. The composite service can be added, for example, to a palette of service definitions within a graphical service composition editor. This is described in more detail using the base protection scenario in Fig 4.

Here three acoustic sensors, a join, and a location computation service are interconnected. This is shown in the upper half of the figure with instances of the services A1, A2, A3, J1 and LL1. A user of the graphical editor may decide that this combination of services and sensors forms a useful, reusable configuration and they would like to group them into a composite service definition named *SniperDetect* that can be added to the palette of available services. In order to do this, a new output from the composite service has to be defined and connected to the appropriate output from the grouped services. This is shown in the figure as the new connection C6 and terminal definition *Sniper Location Output*.

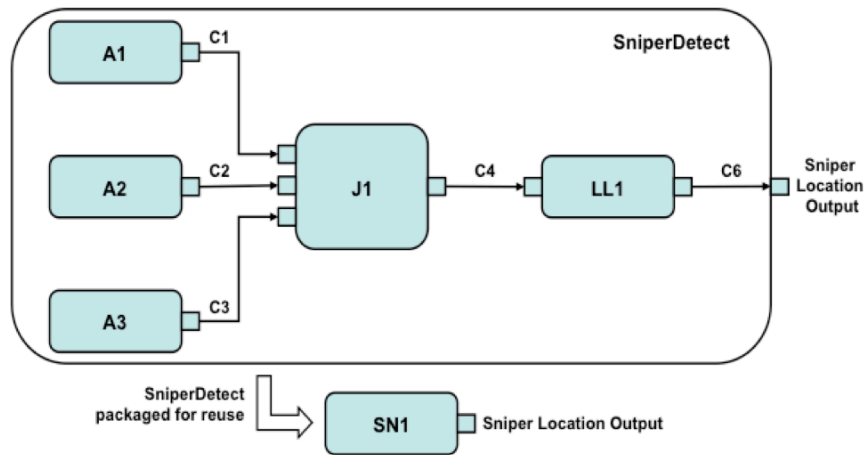


Fig 4: Composite service within the core design model

In CE, composite services are a set of services, connections and terminal definitions.

*conceptualise a ~ composite service ~ CS that is a service and  
 has the service S as ~ service ~ and  
 has the connection C as ~ connection ~ and  
 has the terminal definition TDI as ~ input ~ and  
 has the terminal definition TDO as ~ output ~.*

Using a graphical service composition editor to create a composite service including the SniperDetect service definition, a user would select it from their palette and create a service instance on their workspace; in Fig 5 this is shown as SN1.

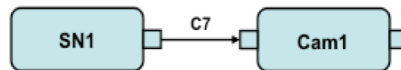


Fig 5: A composite service in a service composition

If the template editor was implemented as a folding editor, then SN1 could be opened to expose the component services from which it has been composed. These composed services are interconnected copies of the service instances originally selected by the user when the SniperDetect service was added to the palette. To differentiate them from another instance of the same composite service, a hierarchic naming scheme is used so that the service instance A1 from the composite service definition is named SN1.A1 in the expanded instance of the sniper service definition. This hierarchic naming scheme applies to composed services nested to any depth.

## 5. AUTOMATING SERVICE COMPOSITION

In the examples provided so far, there are known connections between services that create a data-flow with a certain advanced functionality. This may not be the case in dynamic sensor network environments. Rapid changes in network topologies due to node mobility, bandwidth availability and other constraints can lead to sub-optimal performance of deployed services. Modifying the configuration and placement of services within the network can provide a solution to this problem; we believe this can be achieved through automated dynamic re-composition of services.

Services can be defined in an abstract way and interconnected automatically by matching inputs required by one service with outputs of other services. Accordingly, the abstract definition of a service consists of its inputs, outputs, a mapping from inputs to outputs and the metadata annotations that describe properties of the service:

$$s_i = \{input_i = (input_{i,1}, \dots, input_{i,m}), \\ output_i = (output_{i,1}, \dots, output_{i,k}), \\ f_i(input_i) \rightarrow (output_i, metadata_i(t))\}.$$

Metadata may include such properties as data reliability, cost of a service, energy consumption per output data produced, processing delays, number of other services that use its outputs, etc. The service metadata depends on time (t). Each service may use different types of metadata. A service can only be connected to another service (in a dataflow that describes a composite service) if the service that will provide the information has outputs matching the inputs of the service connected to it.

A *service graph* ( $G_S$ ) can be created from the set of all services in a sensor network environment according to this input/output compatibility:

$$G_S = \{V, E\} \text{ where,} \\ V = \{s_i\} \text{ (one vertex per service) and} \\ E = V \times V, \text{ where } e_{ij} = \begin{cases} 1 & \text{if } output_i \cap input_j \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Service composition is a problem of extracting a subgraph from the service graph which satisfies a certain user-requested set of outputs:  $\Phi = \{output_{\phi,1}, \dots, output_{\phi,n}\}$ . Therefore, the set of services that are chosen should satisfy the user request and both ends of each chosen edge should belong to a composition. Usually, the lowest cost service composition is desired. We model the costs as *communication and processing costs*, which can be represented in terms of energy that is spent to transfer information between services (communication cost) and to run a chosen service (processing cost), for instance. We can formally describe *the service composition problem* (which we proved to be NP-complete) as follows:

For given  $G_S = \{V, E\}$  and  $\Phi$ , find the minimum cost  $V_C \subset V$  and  $E_C \subset E$ , such that:

$$\Phi \subset \bigcup_{V_i \in V_C} (\text{output of } V_i) \text{ and,} \\ \forall V_i \in V_C, (\text{input of } V_i) \subset \bigcup_{V_j \text{ where } e_{j,i} \in E_C} (\text{output of } V_j).$$

We have devised two heuristics that compose a service given a user-request. These solutions differ in the way they traverse the service graph. The *top-down* approach starts with the set of services that immediately provide the user-requested outputs. Once these services are selected, then the algorithm proceeds with finding the services that satisfy the inputs required by the selected services, and so on. Hence the composition process starts from the top of the composition tree (i.e. the set of edges and vertices selected in the service graph), and proceeds downwards. Our second approach, which we call *bottom-up*, starts with the services that require no input. Once these are satisfied (they are inherently satisfied since they do not require the information from any other service), the other services which have all their possible input providers satisfied choose, in turn, their input providers. The bottom-up approach can be seen as a graph scheduling problem, where each vertex can only be chosen once the vertices with an incoming edge to this vertex have been chosen. Clearly, for the bottom-up approach to work, the service graph,  $G_S$ , must be acyclic.

Comparing these two approaches, the top-down approach does not require an acyclic service graph, but on average incurs higher costs and is not always able to compose a service, even though there may be one that is feasible. The bottom-up approach always creates a composition if there is one, is suitable for distributed implementation, and yields low cost compositions. However, it is applicable only to service graphs that are acyclic.

## 6. CONCLUSIONS AND FURTHER WORK

In this paper we have described the bringing together of architectural concepts and development methodologies from the enterprise and shown how they may be adapted to the dynamic, ad-hoc network environment commonly underpinning sensor networks. We have shown how sensor networks, their assets and processing nodes can be represented as services in a service-oriented architecture and modeled using approaches originating from model-driven design principles using a representation, ITA Controlled English (CE), that is both readable by humans and processable by machines. We have also shown how services may be automatically composed through analysis of dataflows from a service graph representation of the available services and assets.

We now return to the fourth trend identified in the introduction; that of situational awareness informing a sensor network user's decision making. Experience gained from other projects originating from the ITA research has shown that CE is rich enough to model complex entities and relationships found in HUMINT reports. We also believe that it can model information from a wide variety of other domains. We have used CE to represent inferencing rules and are actively researching a dialect of CE for querying information sources. These common representations of models lead to an efficient joining of information from what previously were isolated domains. Using a military example, we expect that requests from a commander of the form "I need all available secure video feeds from areas of known insurgent activity" will be supported where the result is a join of information from the sensor network domain, "secure video feeds", and the HUMINT domain, "areas of known insurgent activity", will be readily supported. Thus the use of a common representation such as CE will therefore support coherent, cross-domain situational awareness.

## REFERENCES

- [1] Wright, J., Gibson, C., Bergamaschi, F., Marcus, K., Pressley, R., Verma, G., Whipps, G., "A Dynamic Infrastructure for Interconnecting Disparate ISR/ISTAR Assets (The ITA Sensor Fabric)", Proc. IEEE/ISIF Fusion Conference (2009).
- [2] Ibbotson, J., Chapman, S., Szymanski, B. K., "The Case for an Agile SOA", Proc. Annual Conference of the ITA (2007).
- [3] Ibbotson, J., Gibson, C., Wright, J., Waggett, P., Zerfos, P., Szymanski, B. K., Thornley, D. J., "Sensors as a Service Oriented Architecture: Middleware for Sensor Networks", Proc. 6th International Conference on Intelligent Environments (2010).
- [4] Wright, J., Ibbotson, J., Gibson, C., Braines, D., Klapiscak, T., Geyik, S., Szymanski, B. K., Thornley, D. J., "A Model-Driven Approach to the Construction, Composition and Analysis of Services on Sensor Networks", Proc. Annual Conference of the ITA (2010).
- [5] Mott, D. H., Hendler, J., "Layered Controlled Natural Languages", Proc. Annual Conference of the ITA (2009). [http://usukita.org/papers/5238/ACITA09\\_High\\_Level\\_CNLS\\_06.pdf](http://usukita.org/papers/5238/ACITA09_High_Level_CNLS_06.pdf)
- [6] Sowa, J., "Common Logic Controlled English", <http://www.jfsowa.com/clce/clce07.htm> (2007).

## ACKNOWLEDGEMENT

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.