Third Annual Conference of International Technology Alliance, ACITA'10, London, U.K. September 15-16, 2010

A Model-Driven Approach to the Construction, Composition and Analysis of Services on Sensor Networks

Joel Wright, John Ibbotson, Christopher Gibson, Dave Braines, Thomas Klapiscak,
Sahin Geyik, Boleslaw Szymanski, David Thornley

***Abstract*—This paper investigates the application of model-driven techniques to the construction and composition of services on sensor networks. We present a model that gives the user a visual representation of a service, that can be annotated with semantic information (for example performance characteristics, deployment constraints, policies and rules, etc.) using an appropriate extensible user-oriented vocabulary. We propose the use of UML 2.0 Activity Diagrams as our graphical notation, with semantic annotations represented as properties.**

We show the transformation of the UML model to a semantic representation conforming to an appropriate ontology and use this as the core model for subsequent static and dynamic analysis. We show how the core model can be used to generate domain-specific representations suitable for input to analysis and development tools. Two examples are given: (i) generation of a Performance Evaluation Process Algebra (PEPA) [1] model, and (ii) generation of a specification for the deployment of the service on a sensor network infrastructure. The output of the tools is harvested to provide additional generated semantic information that can be "round tripped" back into the core model, and available for downstream processing.

***Index Terms*—Sensor networks, middleware, fabric, model-driven development.**

I. Introduction

A Sensor network consists of a set of autonomous devices, connected by a communications infrastructure, that collectively monitor and report physical conditions in the real world. To the user, the sensor network offers a set of monitoring services. These services can be augmented with data manipulation (for example filtering and transformation) and data fusion carried out either within the sensor network itself or centrally. Furthermore, sensor events can trigger actions and, again, these might be carried out within the sensor network or centrally.

Manuscript received May 14, 2010. 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.

Joel Wright, John Ibbotson, Christopher Gibson, Dave Braines and Thomas Klapiscak are with the Emerging Technology Services Department of IBM U.K., Hursley, Winchester, U.K. e-mail: {joel.wright, john_ibbotson, gibsoncr, dave_braines, KLAPITOM}@uk.ibm.com

Sahin Geyik and Boleslaw Szymanski are with the Rensselaer Polytechnic Institute. e-mail: {geyiks,szymansk}@cs.rpi.edu

David Thornley is with Imperial College. e-mail: djt@doc.ic.ac.uk

A sensor network such as that described above can be formally modeled in order to describe the architecture and semantics of a solution in the form of a configured sensor network and its component parts. This must describe the architectural structure of the solution, characteristics of the individual building blocks (services and connections), and constraints placed upon the solution by the developer and/or the target environment. Ideally the model must also be independent of any specific implementation and must be presented to the user through a friendly but powerful editing interface.

We propose the use of Model Driven Development [2], based on UML 2.0 Activity Diagrams [3], for describing service construction and composition. Model driven development is a well established design and development paradigm in the enterprise community, and reduces the complexity of solving a problem into an abstract representation of a proposed solution, without burdening the designer with the implementation details for a particular system. Using activity diagrams as the modelling notation has a number of benefits when attempting to design and understand complex systems, primarily that it has a readily understood graphical notation with well defined semantics for capturing behaviour. Data flows can be split, merged, synchronised and processed allowing the designer to express any arbitrary service. The diagrams naturally describe a distributed workflow in terms of autonomous processing and logic elements which lends itself naturally to distributed processing across a network.

We go beyond traditional model-driven development by producing a core semantic representation of our designs, which can then be combined with semantic information from other sources to describe a deployed system, as shown in Figure 1. Using this core representation, possibly in conjunction with domain specific information, we can generate not only service deployments, but also more specialised models including formal models, e.g. process algebra representations, which can be used to further analyse and refine our model of the system as a whole.

In the rest of this paper we discuss the interpretation of UML Activity Diagrams in terms of service composition for sensor networks, how we represent the UML design of a service in our core semantic model, and how this semantic information may be used to generate both a deployment model for a real sensor network as well as a process algebraic model for performance evaluation. Throughout the paper we draw on a simple motivating example in the form of a scan-cue-focus

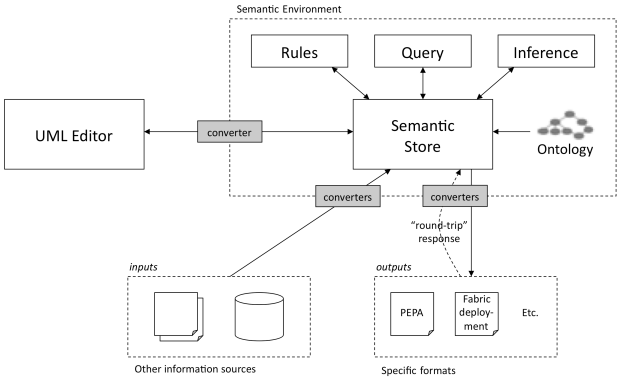

Fig. 1. Model driven service development with a core semantic representation.

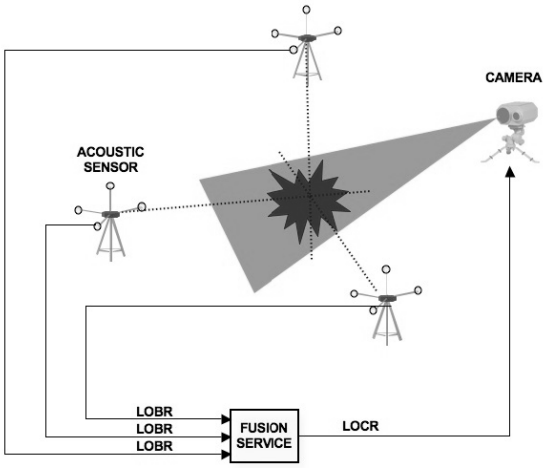

Fig. 2. The base protection scenario.

service which would form one element of a base protection scenario. We conclude by reviewing our progress so far, and outlining ways in which the semantic representation of a service composition may be used in the future.

II. Base Protection Scenario

In this section we describe the motivating example used throughout this paper. Derived from a fielded system, this scenario uses unattended ground sensors to identify, locate and photograph the sources of acoustic events (such as mortar or sniper fire) and make all the sensed data, including the imagery, available to a multiplicity of consumers. The main elements of the scenario are:

- A set of Unattended Ground Sensors (UGS) capable of detecting acoustic events, and generating Line of Bearing Reports (LOBRs) that indicate the bearing from the sensor to the detected event.
- A data fusion service to compute the most likely location of an event from the information contained in two or more corresponding LOBRs. This information is used to generate a corresponding Location Report (LOCR).
- A high-resolution camera, controlled/commanded from an onboard analytics service that, based on the information in a LOCR, calculates the pan, tilt and zoom commands to be issued to the camera in order to direct it towards the origin of the event and photograph the location. The resulting imagery from the camera is made available for consumption by personnel who have registered an interest in receiving this information, for example, personnel at a command and control centre.

We now introduce the graphical notation derived from UML Activity Diagrams that we will be using for modelling service design and composition.

III. Activity Diagrams for Service Composition

Designed as a workflow specification language, activity diagrams can directly represent distributed services on sensor networks by providing a small set of combinators with sufficient expressive capability for modelling and implementing services as data flows. In traditional service oriented architectures workflows described using UML must be translated into a process choreography language (e.g. BPEL [4]) for a central workflow engine. In the context of sensor networks this would be largely unacceptable, both due to the excessive data transfer requirements inherent in this type of system, and the assumed ad-hoc nature of sensor networks meaning that connections to a central management engine would not always be available.

In this section we introduce the various UML combinators available for describing services and service compositions and describe their interpretations in this domain.

Activities

Activities may represent a number of sensor network entities:

- **Data sources -** For example, the unattended ground sensors producing line of bearing reports in our scenario.
- **Data fusion algorithms -** For example, the transformation service producing location reports in our scenario, which run on a network node.
- **Consumers of data -** For example, a consumer of the photographs produced by the camera in our scenario.

Activities may also represent further composite services, however their classification will still fall under one of the three types outlined above. Because we are modelling data flow in the network, the inputs and outputs of all the activities in a service description are explicitly typed and pinned, restricting the activity diagrams to only data flows, as opposed to control flows.

Arcs

Arcs in the context of sensor networks are more than ways of denoting the ordering of activities, they directly represent links between the components of a composition, and as such have attributes that affect the cost and viability

of a composition. These attributes must be considered when producing a service design, as any restrictions placed on the design must be met by the underlying physical links (e.g. a network connection). A single slow link may adversely affect the performance of an entire composition, and in some applications the necessity that a link between activities be secure is an important consideration. Arcs are also used to enforce valid typing in compositions, as an arc will only permit a service input to be joined to a service output of the correct type.

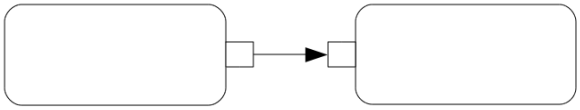

Fig. 3. A UML 2.0 Activity Diagram showing two activities joined by an arc.

Fork

Forking permits concurrent data flows within a composite service, allowing a message to be separately transformed in a number of ways, perhaps to provide a variety of outputs based on a single input, or for independent processing streams before re-composition. Forks signify that concurrent data flows in a service composition are each provided with a copy of an incoming message, a copy being provided to the first processing element of each concurrent flow. Forking on a sensor network would allow not only parallel independent processing streams, but each outgoing edge could represent an individual physical link, thus distributing processing across the network.

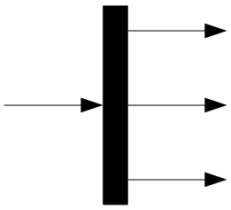

Fig. 4. A UML 2.0 Activity Diagram showing a fork node.

Merge

Merging permits the joining of flows of the same basic type without processing or synchronisation. This operator simply joins two message flows into one, with the number of messages delivered at its end point being the sum of the two (or more) input streams. This type of operator allows multiple sensor streams of the same type to be treated as a single sensor, which would be useful in any application where correspondance between a number of messages is not required (e.g. logging of multiple sensor outputs).

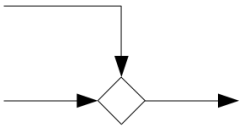

Fig. 5. A UML 2.0 Activity Diagram showing a merge node.

Decision

Decisions mark points at which a service composition may vary its behaviour dependent on the content of a message, e.g. the value of an output from a sensor. A decision point may express explicit message filtering, a message only being passed onto a consumer if certain properties are met, or differing behaviour based on environmental information such as temperature or light levels. In this way decisions may represent messages being delivered to differing recipients based on the content of a message, thus expressing varying behaviour and points at which filtering may occur.

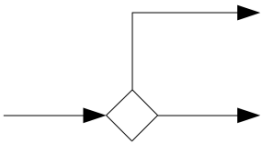

Fig. 6. A UML 2.0 Activity Diagram showing a decision node.

Join

Join nodes represent synchronisation points; a single message containing all relevant information is produced whenever a message has arrived in each of the imputs. The behaviour of a join can be refined according to a join specification, which may define the number of input messages required to produce an output, or criteria for input correspondence. When describing the joining of message flows on a sensor network a join would be responsible for determining when a collection of messages were related, possibly based on timestamps, or identifiers added before a previous fork. Being able to determine related groups of messages will be essential when developing distributed applications.

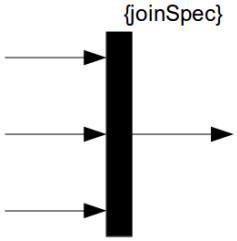

Fig. 7. A UML 2.0 Activity Diagram showing a join node.

Other Activity Diagram Elements

We do not mention here a number of UML Activity Diagram nodes, such as those related to starting and ending activities, and those related to asynchronous signalling, although it is possible that these will be incorporated as design elements in the future. Starting and ending activities is a design element we may wish to consider when representing services with a short lifetime, while asynchronous signalling could represent configuration or command messages to services.

A. Modelling the base Protection Scenario

We now model a subset of the base protection scenario using the diagrammatic elements introduced in this section. We give a UML design for the generation of location report events from UGS line of bearing reports, and show how this simple design of an activity can be connected to sensor feeds to provide a high level service which provides location reports for acoustic events.

The three acoustic sensors each provide a LOBR service that is consumed by a fusion service. In turn this provides a LOCR service that is consumed, in this example, by a camera control service. Collectively this forms a scan-cue-focus scenario, and is an example of service composition on a sensor network. Two steps are required to build this solution:

- **Design - the analysis and design of the composite service:** creation of a composite service template that describes how to build a base protection solution from a set of compatible hardware and software assets.
- **Deployment - the mapping of the design to a set of fielded software and hardware assets:** the instantiation of the solution on a fielded sensor network, if suitable hardware and software assets are available.

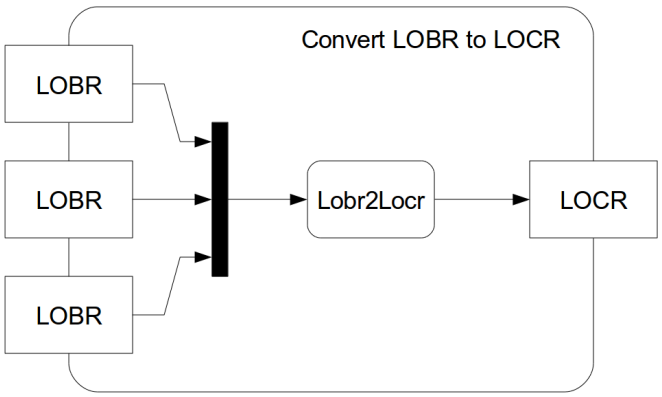

Fig. 8. A UML 2.0 Activity Diagram showing the design of a service to convert three LOBR inputs to a LOCR output.

Figure 8 shows a UML design for a low level software service, *Convert LOBR to LOCR*. This design represents a service which takes three inputs, all of type *LOBR*, and using a join node, passes on sets of inputs which represent a single event. As discussed above, a join specification (e.g. the number of input edges required to receive a message) defining the behaviour of the join itself could also form part of the design. When a set of *LOBR* messages have been received they are delivered to the *Lobr2locr* data transformation, which transforms the data into a location report and produces a single *LOCR* message.

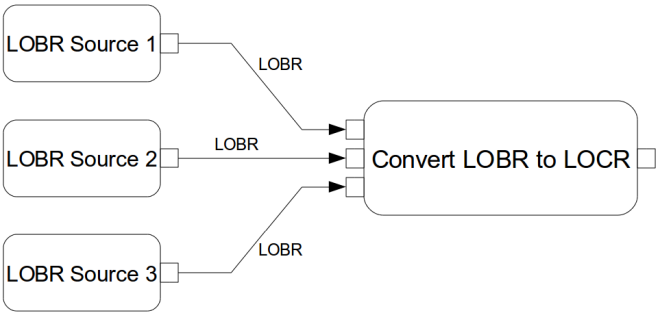

Fig. 9. A UML 2.0 Activity Diagram showing the conversion service connected to appropriate input providers.

High level software services making use of pre-defined actvities such as that described above are also possible, and the heirarchical nature of the model means that the internal details of a service need not be considered when creating compositions. Methods for determining whether a service design can fulfill stated requirements, both in terms of capabilities and quality, are discussed later in this paper.

IV. Semantic Representation of the Service Description

In this section we describe how UML service designs can be represented semantically and how this semantic representation, combined with appropriate annotations and ontologies, can be used to generate two views of the system; a PEPA performance model and a Sensor Fabric [5] deployment model.

A. An Ontology for Service Representation

Figure 10 shows the class hierarchy of an OWL [6] ontology used to describe services on a sensor network. For simplicity we have not shown all the relationships between classes. Here, we have an explicit representation of the various constituents of a deployed service, those being **Connection**, **Port** and **Service** definitions and instances. Connections correspond to the arcs within a service design, whilst Ports represent the service inputs and outputs. Metadata are associated with classes by linking them to members of the Annotation class. This allows a rich description of the model to be produced including information about the types produced or accepted by input or output ports and connection security parameters. Note that this mechanism is distinct from OWL's built-in annotation properties, since these are not usually considered as part of the semantics of the ontology (and thus are largely ignored by reasoners). In addition, other namespaces within the ontology describe components of a deployed sensor network including the Node, Platform, Sensor and Feed components together with the end users (Actors) and the tasks they perform.

Service definitions may also contain instances of other composite services representing a hierarchic service description. The ontology elements, service and port definitions and

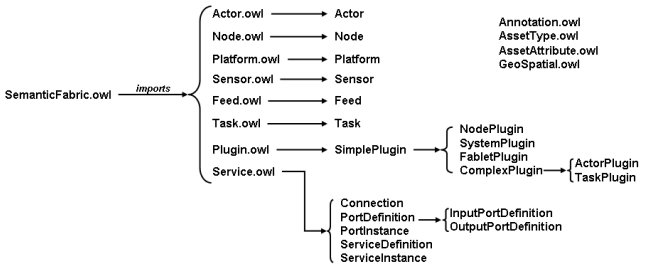

Fig. 10. An Ontology for a Sensor Network.

instances together with connections, provide the anchor points for design annotations supporting the UML activity model for sensor network designs. The ontology allows for annotations, providing support for application into arbitrary domains. Note that the types of information produced by the sensors in a service design have a general type which can be taken from any sensor ontology. Initially we expect the typing information on these sensors to come from the ISTAR ontology produced by Project 8 as part of their work on sensor mission matching [7], however alternatives such as SensorML [8] or OntoSensor [9] may also provide input.

The use of OWL as the core semantic modelling language of Fabric services has three primary motivations:

- OWL is an open standard recommended by the W3C, meaning that the data produced by our system will be inherently and readily reusable, making integration and interoperation with other existing technologies more straightforward. For example, we have the opportunity to make use of the aforementioned ISTAR ontology developed as part of Project 8 in the ITA programme.
- A wide range of freely available tools have been developed aimed at helping developers and end-users of OWL-based technology. Of particular interest are generic visualisation and exploration tools such as Protege [10], and libraries such as the OWL-API [11].
- Owing to its formal underpinnings in Description Logics, the semantics of OWL are well understood and it has been proven that decidable and complete reasoning algorithms exist for inferring new information from all but the most expressive species of OWL ontology. Moreover, a number of general reasoner implementations exist, many of which are under active development, such as Pellet [12], Jena [13] and HermiT [14]. Making use of technologies such as these will allow us to sidestep the considerable overhead and risk of developing a custom reasoner from the ground up.

The final point above is of particular interest and merits further discussion. Access to implicit information contained within an ontology can offer a number of benefits in the context of running services on the sensor network, some examples of which are given below:

- Errors in a service instantiation can be detected at design-time, potentially speeding up and easing the development of demonstrably robust services. A simple example of this might be checking that connections between nodes go from input to output pin, but not visa versa. More complex design-time checks may involve ensuring that proposed sensor configurations lie within defined manufacturer thresholds - for instance, that a complementary group of acoustic LOB sensors are located within range of one another.
- Implicit information can aid with the process of automated service composition and discovery. A very simple example would be inferring the potential compatibility of two services with synactically different but semantically equivalent output/input types (e.g. ”Text” and ”String”) as determined by some inference. A more complex matchmaking service might employ subsumption reasoning to allow for the querying of compatible services according to some specification of required functional and non-functional parameters.
- Degrees of trust held by actors in the system can be propagated through a network of connected services using transitive properties or property chains. Similar techniques could also be used to infer the overall level of security of a candidate flow through the system (i.e. a flow is only as secure as its weakest link).

V. Generated Models

In this section we describe the generation of useful models from the core semantic representation of service designs. We begin by describing our interface with the ITA Mission Abstraction, Requirements, Structure (MARS) framework [15], which makes use of a transformation of our core semantic model describing a service composition into PEPA. The conclusion of this section describes the mapping of our service designs onto a deployed sensor network system in the form of the Fabric.

A. Interfacing with MARS

The MARS framework formulates and quantifies Measures of Merit (MoM) that explain the performance and effectiveness of systems for command and control support by creating computational links between the details of an information system deployment and the satisfaction of command requirements. The approach taken to the analysis of composite systems is compatible with the NATO COBP for C2 analysis, which describes a complete route from deployments on the ground to the requirements of command. The range of measures being developed in the MARS programme includes basic accuracy and delay with decay characteristics computed on the mission design [16], through contextualized emergent utility features [17], [18], to higher level reasoning in support of balance of investment decisions relating to technology development and deployment [19].

Specifically, a service is treated as a system of systems, each of which is analysed in terms of performance characteristics in an activity that can provide support to getting Measures of Performance (MoP). The whole system is analysed for effectiveness in solving a problem, giving Measures of Effectiveness (MoE), by relating the outcomes in the problem space to the performance in the solution method space. An MoP for

a sensor network might characterize the bounds on delay between event and notification, and bounds on the capacity of the network to deliver notifications, whereas a corresponding MoE might give the percentage of events for which the combination of notification and subsequent requests and responses result in the satisfaction of a command requirement.

To predict the performance of a composite service, it is necessary to relate the required outputs to the inputs and other influences in a stochastic model that can explain the timing and charateristics that emerge from the combination of interacting or contributing services. In a service oriented architecture, services are constructed at runtime from novel combinations of fielded services. This naturally motivates the use of a componentized approach to performance analysis, as this supports simplification of the model construction, formation of a library of services for which knowledge and experience can be built, and the potential for re-use of results calculated for individual services to increase efficiency. The theoretical work and concrete numerical computation are linked by the use of the Performance Analysis Process Algebra (PEPA).

Once defined, the semantic representation of our service descriptions can be transformed into PEPA, which the MARS framework can use to form components of larger models describing mission scenarios. The initial generation of a PEPA model describing a service composition requires three pieces of information derived from the semantic model:

- UML Node types.
- Node connection information.
- Rate information (either explicitly stated or derived).

Previous work on transforming UML Activity diagrams [20] requires each element of the diagram itself to be tagged with rate information, and acts as a direct translation of the diagram itself. Here we can use the information stored in a semantic repository, e.g. the speed of network links or available processing power on nodes, in order to automatically derive rate information. We can also refine our performance model, and use it to inform deployment, because the effects of a particular proposed deployment could be added to a model, taking into account such things as network contention and multi-hop links.

Figure 11 shows a fragment of the PEPA code generated from the UML depicted in Figure 8. Note that here we only consider the simple case where all three corresponding messages arrive. More complex behaviours, such as allowing the process to continue when only a subset of messages arrive, are the subject of further research.

For SOA performance analysis, MARS modelling provides the interface for which a PEPA component may be produced from the SOA definition framework, e.g. in [16] the quality of a sensing service providing detection information. The performance of the system is expressed in the delay and error of detection reports delivered to the customer. The utility is expressed in the correction of the customer's situational awareness (i.e. solving his problem). PEPA is used because it naturally describes support with its use of cooperation over actions.

$$
\begin{array}{rcl}
Lobr2Locr & \stackrel{def}{=} & (lobr_1, \top).Lobr2Locr_{100} \\
 & + & (lobr_2, \top).Lobr2Locr_{010} \\
 & + & (lobr_3, \top).Lobr2Locr_{001} \\
 & & \\
Lobr2Locr_{100} & \stackrel{def}{=} & (lobr_2, \top).Lobr2Locr_{110} \\
 & + & (lobr_3, \top).Lobr2Locr_{101} \\
 & & \\
 & \vdots & \\
 & & \\
Lobr2Locr_{111} & \stackrel{def}{=} & (lobr_{set}, r_1).Lobr2Locr \\
 & & \\
Lobr2Locr_{act} & \stackrel{def}{=} & (lobr_{set}, \top).(locr, r_2).Lobr2Locr_{act}
\end{array}
$$

Fig. 11. PEPA Model Fragment of Convert LOBR 2 LOCR.

B. Mapping from UML Semantic Representation to a Fabric Deployment

In this section we describe how to generate a deployment model for the ITA Sensor Fabric (Fabric) from the semantic representation of a service design. Services on the Fabric do not follow the conventional view of a service requester invoking a service provider which responds to the invocation. This request/response service model is widely used in Web Service applications but is not appropriate for the stream-based services found in an event based messaging environment such as the Fabric for reasons discussed in Section III. Note that we do not give full deployment details, but rather describe how the elements of a design relate to Fabric concepts.

The Fabric is an extensible middleware platform that spans the network from the data centre 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 maximising its availability and utility to applications and users. A plug-in architecture allows new functions (such as filters, transformations, policies, security, and event detection algorithms) to be deployed into the sensor network and selectively applied to sensor messages en route to the user.

A Fabric node is a node on the sensor network that runs an instance a message broker, a Fabric Manager, and a Fabric Registry. As the main Fabric service running on a node, the Fabric Manager has wide ranging responsibilities, building the major capabilities of the Fabric on top of the message broker, which provides the actual communications infrastructure, and the Fabric Registry, which is responsible for storing the configuration and operational status of the Fabric infrastructure. The major features and functionality provided by the Fabric Manager include establishing the communication channels between nodes, tracking the operational status of connected sensors and nodes, and registering local data fusion algorithms as intelligence assets with the Fabric Registry. However, most importantly from the perspective of service design and deployment, the Fabric also provides a container for running software services which may perform in-network information fusion, filtering, and other algorithms, as well

as a point from which the capabilities of the Fabric may be extended. For more information regarding the ITA Sensor Fabric see [5].

1) UML Activity Diagrams and the Fabric: As we described in section III, activities in a UML service design may represent a number of entities in a sensor network, and hence a Sensor Fabric deployment; in the context of the Fabric, activities may represent sensors producing data, actors consuming data, or algorithms running on Fabric Nodes. Here we describe the mapping of the various UML Activity diagram entities onto an instance of the Fabric.

Join, merge and decision points are special cases of software services which perform those roles without implementing any form of data transformation. Forking a flow in the Fabric simply represents two activities subscribing to the same data feed, and thus does not require an explicit implementation; the Fabric includes the concept of message identification to allow a resultant join to identify corresponding events if necessary.

Arcs represent a producer of information publishing, and a consumer subscribing, on a single Fabric topic; data may pass through multiple Fabric Nodes on this journey. Links only join outputs to inputs with the corresponding type so that the message provided as input to a data transformation or end subscriber contains data in the expected form. This enforcement is performed using information about sensors and services contained in the Fabric Registry and semantic store.

By viewing all services as activities we allow heirarchical definitions, with component activities viewed as data transformation black boxes which may be distributed on the network themselves. This fits with the 5-box schema view of the services introduced in [21], and allows higher level composites to be tagged with semantic information (not just the lowest-level simple services).

VI. Conclusions

A UML model used to design high level services allows compositions to be specified in a clearly understood and graphical manner, whilst having a well defined meaning. A corresponding semantic model of composition not only reduces the complexity of producing a concrete solution, but due to the high-level abstract representation, can be used to inform the recomposition or reconfiguration of a service in the event of a network topology change or network failure, a common event in ad-hoc wireless networks.

By representing the service design semantically we allow generic transformations into a variety of formal and deployment models. These transformations can be extended into new and unknown models and domains without modification of the model due to the semantic representation not being tailored to any particular use or analysis; we simply need any required information, beyond the details of nodes and connections, to be contained in, or derivable from, the semantic store. Also, existing tooling and techniques for transforming UML are still available to us because the original UML can be recreated from our semantic model. A semantic representation is therefore far more flexible and extensible in terms of both its sources of information and its potential outputs.

VII. Future Work

In this section we lay out further directions for research based on the semantic model representing service designs in conjunction with a sensor network.

A. Round tripping

We expect transformations into formal models to be used as part of an initial design process prior to deployment in the sensor network. Results from the analysis of the formal model may then be used to enhance or modify the annotations on the design model. The flexibility of semantically annotating the model allows us to easily integrate the results back into the original UML model as feedback to the designer/developer. Due to our use of a semantic store to represent the sensor network and its services, the results of any analysis are made available as potential inputs to any further reasoning. We describe this as round-tripping.

Semantic annotations can also be used to capture physical properties of a deployed sensor network. This supports the second, concrete, round-tripping route show in Fig 12 between the design model and the services deployed on an active sensor network. The Fabric provides instrumentation on the sensor network allowing real-time metrics to be gathered and a profile of the deployed network performance to be generated. This profile can be used to provide further annotations within the design model to allow subsequent refinements of the formal representation that will be more suitable for the deployment environment.

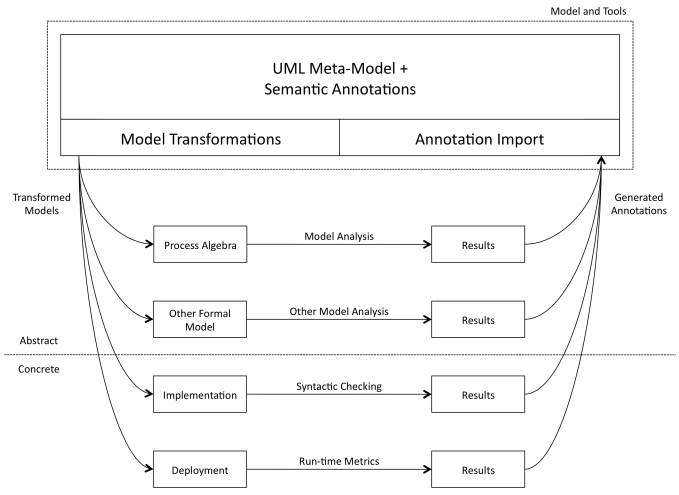

Fig. 12. Analysis and Model Round Tripping.

We expect the UML design model to undergo multiple round-trips between the formal and deployed states during its lifecycle. By evaluating potential services in this way and using the information gathered through round tripping we can choose the deployment which best serves our needs, e.g. we could choose the deployment that has the least runtime impact on the services running at the time of deployment, or decide the best way to fulfil any particular service requirement. In one specific instance, by round-tripping through the MARS

framework the semantic repository can include perfomance assessments that may be used to inform future composition choices.

B. Other Transformations

We must investigate further transformations of the semantic model into alternate formal models for analysis, but it is important to consider other uses of our service designs, e.g. generating skeleton code to implement a design. Code generation in this context is non-trivial because we must decide when to use an existing service and when to generate a code skeleton. Another important consideration is the most appropriate method for generating code so that it may be distributed across the network. Even the decision of when to use an existing service is a potential source of problems because, in order to generate only the required code, we would need design time access to a list of available services.

REFERENCES

[1] J. Hillston, "A Compositional Approach to Performance Modelling," 1996.

[2] J. A. Clark and J. L. Jacob, "Model-Driven Development," *IEEE Software*, vol. 20, pp. 14–18, 2003.

[3] *Unified Modeling Language (UML). Specification ISO/IEC 19501*, OMG Std., Rev. v2.1.2.

[4] *Web Services Business Process Execution Language*, OASIS Std., Rev. v2.0. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[5] J. Wright, C. Gibson, F. Bergamaschi, K. Marcus, R. Pressley, G. Verma, and G. Whipps, "A Dynamic Infrastructure for Interconnecting Disparate ISR/ISTAR Assets (the ITA Sensor Fabric)," in *Proceedings of the 12th International Conference on Information Fusion*, 2009.

[6] D. L. McGuinness and F. van Harmelen, "OWL web ontology language overview," World Wide Web Consortium, W3C Recommendation, 2004. [Online]. Available: http://www.w3.org/TR/owl-features

[7] M. Gomez, A. Preece, M. P. Johnson, G. de Mel, W. Vasconcelos, C. Gibson, A. Bar-Noy, K. Borowiecki, T. L. Porta, D. Pizzocaro, H. Rowaihy, G. Pearson, and T. Pham, "An Ontology-Centric Approach to Sensor-Mission Assignment," in *Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management*, 2008.

[8] *Sensor Model Language (SensorML) for In-Situ and Remote Sensors. Specification OGC 07-000*, OGC Std., Rev. v1.0. [Online]. Available: http://vast.uah.edu/downloads/sensorML/v1.0/specification/07-000_SensorML_Implementation_Specification.pdf

[9] D. J. Russomanno, C. R. Kothari, and O. A. Thomas, "Building a Sensor Ontology: A Practical Approach Leveraging ISO and OGC Models," in *ISO and OGC Models,The 2005 International Conference on Artificial Intelligence, Las Vegas, NV, 2005*. Press, 2005, pp. 637–643.

[10] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen, "The Protg OWL Plugin: An Open Development Environment for Semantic Web Applications," in *International Semantic Web Conference*. Springer, 2004, pp. 229–243.

[11] M. Horridge and S. Bechhofer, "The OWL API: A Java API for Working with OWL 2 Ontologies," in *Proceedings of The OWL: Experiences and Direction (OWLED) Workshop*, 2009.

[12] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, June 2007. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2007.03.004

[13] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations." in *Proceedings of the 13th international conference on World Wide Web - Alternate Track Papers & Posters, WWW 2004, New York, NY, USA, May 17-20, 2004*, S. I. Feldman, M. Uretsky, M. Najork, and C. E. WillsSemanti, Eds., 2004, pp. 74–83.

[14] B. Motik, R. Shearer, and I. Horrocks, "Hypertableau Reasoning for Description Logics," *Journal of Artificial Intelligence Research*, vol. 36, pp. 165–228, 2009.

[15] D. Thornley, R. Young, and J. Richardson, "Development of a Mission Abstraction Requirements Structure (MARS) and stochastic modelling for sensing service-driven mission performance prediction," Department of Computing, Imperial College London, Tech. Rep., August 2008, 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. [Online]. Available: http://pubs.doc.ic.ac.uk/ita-mars-tech-report-v09/

[16] D. J. Thornley, C. Bisdikian, and D. F. Gillies, "Using stochastic process algebra models to estimate the quality of information in military sensor networks," in *Proceedings of Modeling and Simulation for Military Operations III, SPIE DSS 6965*, 2008.

[17] D. J. Thornley, R. I. Young, and J. P. Richardson, "Toward mission-specific service utility estimation using analytic stochastic process models," in *Proceedings of SPIE Defense, Security and Sensing 7352A*, 2009.

[18] D. J. Thornley, D. F. Gillies, and C. Bisdikian, "A stochastic process algebraic abstraction of detection evidence fusion in tactical sensor networks," in *Proceedings of SPIE Defense, Security and Sensing 7348*, 2009.

[19] D. J. Thornley, D. F. Dean, and J. C. Kirk, "Warfighter decision making performance analysis as an investment priority driver," in *Proceedings of Modeling and Simulation for Defense Systems and Applications V, SPIE DSS 7705*, 2010.

[20] M. Tribastone and S. Gilmore, "Automatic Extraction of PEPA Performance Models from UML Activity Diagrams Annotated with the MARTE Profile," in *Proceedings of the 7th International Workshop on Software and Performance*, 2008.

[21] J. Ibbotson, C. Gibson, J. Wright, P. Waggett, P. Zerfos, B. K. Szymanski, and D. J. Thornley, "Sensors as a Service Oriented Architecture: Middleware for Sensor Networks," in *Proceedings of the 6th International Conference on Intelligent Environments - IE'10*, 2010, to appear.