Senaps: A platform for integrating time-series with modelling systems

Mac Coombe [a], Paul Neumeyer [b], Joe Pasanen [a], Chris Peters [a], Chris Sharman [a], Peter Taylor [a]

[a] CSIRO Data61, Hobart
[b] University of Tasmania, Hobart
Email: peter.taylor@csiro.au

Abstract: Models of physical systems are the foundation of many scientific and decision support systems. These models rely heavily on observational data, typically collected from sensors. Increasingly this data comes from a wide range of sources. For example, agricultural models often require data from climate observations, soil conditions, on-farm equipment, seasonal forecasts, among others. Integration of these data with models is very time-consuming and often is repetitious across different models. Furthermore, automation of model runs is difficult due to the complexity of managing data dependencies.

We have developed a distributed system, Senaps, to support automation of sensor data retrieval and coupling with model execution in a scalable way. It has been developed over many years across scientific disciplines, including water management, agriculture, aquaculture, and related Information, Communication and Technologies areas. It has been used, and is in use, by a diverse range of projects, resulting in a flexible system that is not tied to a specific domain.

Senaps includes a publish-subscribe subsystem that handles ingestion of disparate time-series data. It supports stream processing, such as quality assurance and data checking, and automates data ingestion with monitoring and recovery. The storage and access subsystem is a scalable time-series backend with an Application Programming Interface (API) to allow third party developers to build on. It has a range of features including dynamic temporal aggregations; fine-grained access control to support data privacy and sharing (users can elect to share data between organisations); metadata for sensor data management; and controlled vocabularies.

The focus of this paper is the model integration subsystem, which provides the model integration and automation features. This system builds on developments in cloud and container-based computing to isolate a user submitted model's runtime environment and provide access to the data backend. APIs are provided to handle environment images (e.g. Linux with R), model definition, workflows (instances of a model), and running of model jobs.

We have successfully used this system to automate model runs and provide continuous results from a number of parameterised models. We have hosted a number of models on the platform, including a timber drying model and two agricultural prediction models. Being tied to a robust sensor-data backend ensures models are run on the most recent data and removes the need for model developers to continuously manage model execution. Results from the model are automatically available and can be easily shared between users and organisations. In this paper, we detail the technical challenges in implementation, provide example results from a running model, and describe our next research steps.

Keywords: *Sensor data platform, Internet of things, sensor model integration*

1. INTRODUCTION

Making observations is the basis of many sciences. The increasing sophistication of sensing devices is improving our observing capability, and our understanding of the physical world. Much modern science involves the combination of sensor data (understanding current state) and simulations (capturing the nature of observed processes). Additionally, science is become more cross-disciplinary.

This has resulted in challenges in the way we manage and share sensor data. For example, agriculture science has long used sensor data from farms to understand current conditions. Weather observations, short term forecast, seasonal forecasts, and river observations are now also used. We must include these data in ways that make sense in the context of specific domains and challenges. This requires us to link observational and sensor data to our physical models.

This introduces many challenges. How do we integrate data from different sensor sources? How do we access third party data, such as from meteorological organisations? How do we control access to sensitive sensor data? How do we automate our systems to allow us to focus on science? How do we avoid repeating this for every research project we are conducting? These challenges are the context that initiated our work on the Senaps platform.

2. PROBLEM DOMAIN

Digiscape is a CSIRO initiative (CSIRO 2016) that is researching the next generation of agricultural support systems. It is doing this by bringing experts from agriculture, climate science, land and water management, and ICT. The goal is development of predictive systems used in industry to improve practices. The research areas include agricultural yield prediction, aquaculture management, and greenhouse gas markets. Crosscutting activities include climate data, data uncertainty, and data platforms.

The Digiscape activities rely on access to up to date, relevant and accessible data. Senaps is an important part of the data platform that will enable this research. The research projects need access to near real-time data streams from sensors. They need climate data in a range of forms. They need to be able to run parameterised models across thousands of locations.

Sense-T (University of Tasmania 2017b) is a partnership between the University of Tasmania, CSIRO Data61, Tasmanian Government and Australian Government. Sense-T is an industry based research and innovation program showing how real-time sensor data can be used to enhance industry sectors across Tasmania. Sense-T has over 20 projects across multiple industry sectors. The Sense-T Data Platform (an instance of Senaps) provides a central data management and analysis platform.

The Pasture Productivity project (M.T. Harrison et al. 2017) uses the Analysis Service of Senaps to orchestrate the execution of the DairyMod (Johnson et al. 2008) dairy pasture growth model. This model is developed using Delphi, a derivative of Pascal. The project objective is to develop easy to use tools online tools for dairy farmers to help predict pasture growth. The Senaps Analysis Service has been used to automate model runs within this project.

The Forest and Wood Products project adapted a model originally designed to model the drying of timber in a kiln environment and applied it to a yard drying environment. The model was extracted from a C++ program as part of the Clever Kiln Controller (CKC) (Innes 1996), a Windows application, and was hosted on the Analysis Service using a light weight python wrapper to internally call the adapted C++ binary. A bespoke client application (University of Tasmania 2017a) was created for the project to run the yard timber drying model on demand via the Analysis Service API with custom parameters.

3. RELATED WORK

Sensor data management systems, and more recently Internet of Things (IoT) systems, have a long research history. These systems also carry different names, depending on the perspective of the research, including Sensor Webs (Delin and Jackson 2001), Wireless Sensor Networks (Akyildiz and Kasimoglu 2004), Environmental Sensor Networks (Hart and Martinez 2006), Semantic Sensor Networks (Kerry Taylor & Arun Ayyagari 2006), Semantic Sensor Web (Sheth, Henson, and Sahoo 2008), Sensor Web Enablement (Bröring et al. 2011), among others. These systems typically address challenges at different levels of the communications stack: lower-level systems looking at sensor-sensor communication or sensor-gateway communication, up to high-level systems, such as the semantic and web-service based systems. A full review and comparison with existing systems is not in scope of this paper. However, the Senaps system has drawn on aspects of much existing work, which we highlight here.

Senaps addresses the need of a sensor data aggregator. It is able to stream data from relatively low-level communication frameworks to higher level systems such as File Transfer Protocol (FTP), other APIs, and web services. It has been driven by the need of researchers to quickly access data from sensors, and other observational data systems, in a consistent and easy way. The Senaps time-series engine (described below) has a controlled vocabulary sub-system, leveraging much of the work in the vocabulary and Linked Data research arena (S. J. Cox, Yu, and Rankine 2014), and some specific time-series standards (Taylor 2012, 2). The underlying message encoding is inspired by the Observations & Measurements information model (S. Cox 2010). Aspects of the original design of the platform made use of concepts from Sensor Web Enablement (Bröring et al. 2011) and the suite of related standards from OGC (Botts et al. 2008). This included a number of CSIRO projects using these systems and standards (Hugo et al. 2011; Terhorst et al. 2011).

The Senaps Analysis Service provides a time-series analytics capability to the platform focusing on hosting third party models interfaced through python or R scripts, while adhering to the Senaps controlled vocabulary sub-system and metadata model. The Analysis Service combines concepts such as serverless architecture (Fowler 2016), application containerisation, and workflow patterns (Workflow Patterns Initiative 2017) to provide an integrated solution via an API. There are already products in this space who provide the building blocks to achieve such a system. These include AWS Lambda, Google Cloud Functions, Azure Functions in the serverless space; Kubernetes, Docker in the containerisation space; and Kepler, Workspace, Tervana, and Pegasus in the workflow space. While these systems may be used in our ultimate implementation, the domain specifics around integration of spatio-temporal data and models will need addressing. This was also the finding of the recent work developing a framework for graph processing of astronomical data (Wu et al. 2017).

4. SYSTEM DESIGN

Senaps has been designed as a loosely coupled set of components that interact through either event-based message passing or APIs. A high-level overview of the system is shown in **Figure 1**. The stream processing and ingestion component uses the publish-subscribe pattern to handle events generated from various sensor (and other) sources. There is a distinction made in the system between time-series and gridded data. This represents common data management practice within environmental data communities and allows us to make use of open source tools for gridded data, specifically THREDDS (John Caron and Davis 2006).

The following sections detail the individual component's design and implementation.

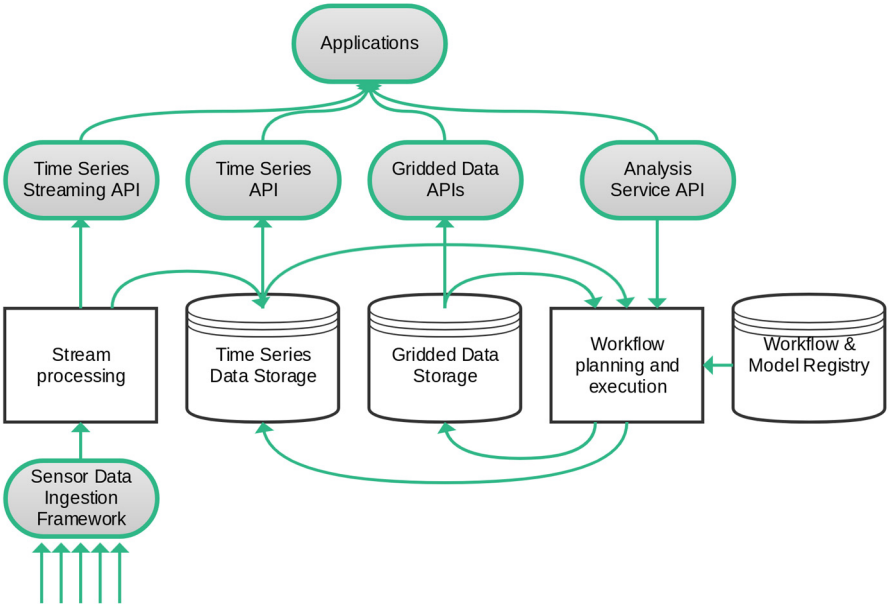

Figure 1. System overview

4.1. Ingestion and storage sub-system

Near real-time data streams are ingested into the Senaps platform via an internally developed software framework called Sensor Messaging Gateway (SMG). SMG provides a library of data sources and streaming processors for continuously parsing, transforming and processing incoming data in near real-time. It is similar

in nature to Apache Flink (and related streaming projects) in that it a distributed data streaming engine. SMG focusses on time-series, of varying value-types, and provides specific features that deal with time-series processing. SMG nodes have a Source-Processing-Sink structure that allow arbitrary topologies to distribute messages through distributed systems. The standard setup is a Source to message broker, to data storage layer pattern; for example, sensor sources to RabbitMQ to MongoDB. The system supports time-window caching and processing, allowing stream-type processing to be written for pre-and-post processing of data, such as deriving new time-series, quality assurance and control etc. It also supports guaranteed message delivery through RabbitMQ's acknowledgment framework.

The platform includes a time series storage component which stores data streams comprised of key value pairs of timestamp and a data value. The data value can be one of four distinct types; geospatial trajectories, scalar, vectors and images.

Time series data is stored in a MongoDB document store. Individual time series samples are aggregated into documents. The aggregation period is based on the document size rather than a fixed temporal period to ensure average document size invariant to sample rate, this in turn ensures index sizes remain manageable and predictable regardless of data sample rate. A common dataset, the SILO Patched Point Dataset (DSITI 2017) comprises of approximately 150,000 individual scalar time series where each time series contains over 120 years of daily samples. This equates to 6.5 billion sample points. Another example dataset records a 20Hz plethysmographic waveform used to calculate heart rate of Oysters. The time series storage component provides high performance random access to temporal slices to any data stream via its API. The details of data storage structure, comparisons with other solutions and performance evaluation of the time series components are beyond the scope of this paper.

4.2. Analysis service design

The Analysis Service component of the Senaps platform is designed around a concept of "operators": reusable and configurable software components which are used to perform processing of time-series and/or gridded data. Each operator exposes a number of "ports", which serve as placeholders for input and output parameters as shown in Figure 2. Depending on the operator's needs, each port may accept either one or more Senaps data streams, a single THREDDS dataset, or a plain-text configuration document.

Operators are applied to a particular processing task using "workflows". A workflow realises the operator's inputs and outputs as "data nodes", which bind specific data streams, gridded datasets or text documents to the operator's ports. Workflows are executed by creating "jobs", which can be created on demand or can be scheduled to occur repeatedly on an ongoing basis. Scheduling of jobs is achieved using a syntax to specify time-intervals for execution.

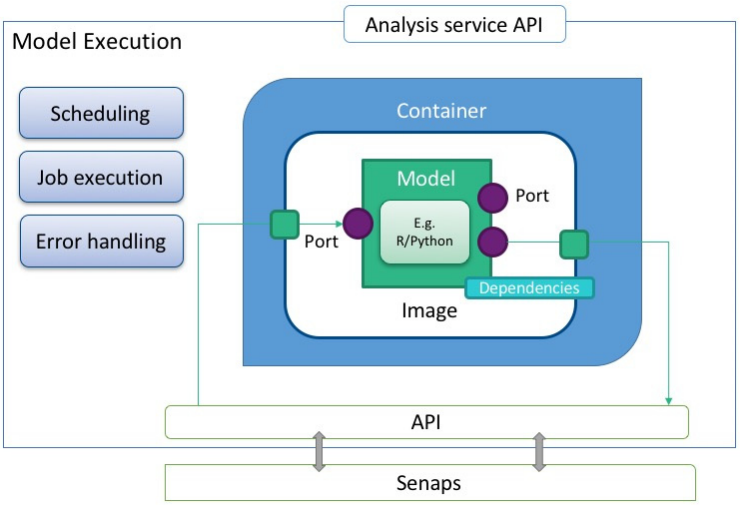

Figure 2. Model execution system overview

A key feature of the system is the ability of end-users to implement bespoke operators known as "models". These consist of code written in the Python or R programming languages, and can be used to encapsulate custom computational models or to implement pre-processing or post-processing facilities. Because all models ultimately execute in a Docker container the model developer can also include data files and native executables.

This allows embedding pre-existing models developed in any language which will compile to the native architecture of the Docker host and run under the Docker container base image. All models must extend a base image which is managed by administrators. The base images contain operating system libraries and typically represent the majority of the data volume for a model. However, Docker uses a union file system to avoid duplication on disk. Even though end-user models are based on a full Ubuntu distribution and include run-times such as Python and R and common libraries each new model only consumes disk space required by the *difference* between the model image and its base image. This allows for scalability of model storage.

Internally, each model container exposes a common API. This API is used by the execution system to control models. An implementation of this API for each base image provides entry points and callbacks that the model developer can use to integrate their code. Currently the system has implementations with bindings for Python and R. Since the Model container interface is specified as a API, any new direct language bindings are possible. The Analysis Service is exposed to external clients via a API, using the Hypermedia Application Language design pattern. This service allows for creation and discovery of models, workflows, schedules and jobs.

When an execution job is created - either on demand or on a schedule - it is placed into a task queue in a pending state, awaiting whatever execution resources are required in order for it to execute. This is shown in **Figure 3**. When the required resources become available, the workflow is executed by supplying the bound data nodes to the encapsulated operator via its ports. The operator executes, and supplies its outputs by modifying the data nodes bound to its output ports. The status of the job (i.e. pending, executing, successful or failed) may be queried at any time though the API. Upon successful completion, the job's representation in the API is updated to show the modified values of the output data nodes, and to include execution statistics such as overall run time. In the event the job failing, the output data nodes remain unchanged and a stack trace is returned to enable diagnosis of the problem.

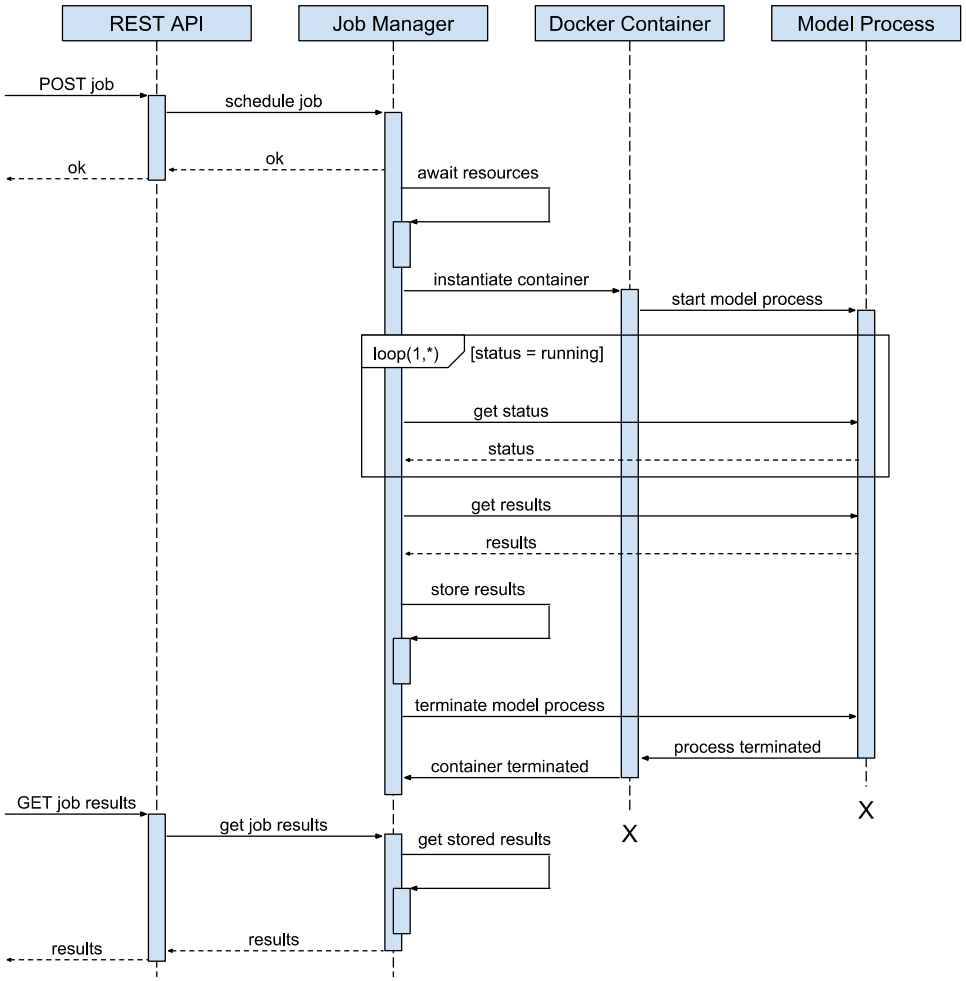

Figure 3. Analysis service lifecycle

Implementing a new model consists of uploading its source code to the API, as well as a manifest file. The manifest file supplies metadata about the model (such as a description, author, and version), declares the input

and output ports that the model supports, and declares any third-party dependencies required for the model to run. The Analysis Service uses this information to build a Docker image containing the model code, a suite of libraries for interacting with the Senaps platform, and the other third-party dependencies declared in the manifest. This approach offers benefits for repeatability, scalability and security. Packaging the code and its dependencies together when the model is first uploaded ensures that the model always uses consistent versions of its dependencies, and does not fail if a dependency becomes unavailable for installation at a later date. Since Docker images are self-contained and portable, we are also able to execute models on different host machines at runtime, leading to both robustness in the face of hardware failure, and to the ability to run multiple instances of the model in parallel. Furthermore, since the model's dependencies are only resolved and downloaded when the model is first installed, we are able to restrict internet access at runtime, leading to reduced attack surface for malicious models.

Model development and testing is supported through two mechanisms. First, the Python libraries used by models to interact with the service also contain utility methods for creating a mock execution environment. This can be used to support offline testing of models before they are uploaded to the Analysis Service. Second, the service implements a debugging mode for models which returns a detailed log of the model's execution. This facility supports the Python native logging framework, allowing models (and any dependencies which use the logging framework) to produce log messages in an idiomatic manner.

The Analysis Service uses the same organisation- and group-based permissions system employed by the time series storage component. In practice, this means that models and workflows may be restricted to access by personnel within a single organisation, or within an organisational group. The ability to share items within organisational groups with other organisations means a model author can share a read only view of their model while having full control over access. This allows receiving organisations to create workflows based on the shared model enabling collaboration between organisations.

5. DISCUSSION AND FUTURE WORK

The Senaps platform is actively being used in a range of research projects from different domains. While we have not included quantitative performance results within this paper, our internal testing shows that the platform has effectively scaled to handle billions of time-series values without sacrificing data access latency. The aggregation framework for time-series has allowed users to quickly visualise dense, long period time-series in a range of research applications. There are some current and foreseeable challenges we will be focusing on, these are outlined here.

It is often the case that once models are setup and validated, operators will want to run instances across many parameter sets. These parameters may represent different locations, time-ranges, input parameter ranges, for example. We are developing approaches to express these setups without having to enumerate every instance of the input set. This would allow run-time queries such as 'run my model at every Bureau weather station in the country', without having to manually specify sites, or rely on external - and thus brittle - scripts. This mechanism will also support distribution of parallel runs to provide scalability.

Having model processes that are dependent on one another is a desirable feature, especially when generalised models are available that can be reused. For example, having a downscaling algorithm that precedes inputs into another model is a common pattern in modeling systems. If a dependency graph is maintained it is possible to reproduce outputs when upstream data or models are updated.

We have not significantly tested scaling up of the model runs within the analysis service. Running the models in highly parameterised mode will require this for reasonable run-times. There is a range of options, from running parallels tasks on grids to high-performance computing environments that use job queuing for compute sharing. We are actively investigating options and will be testing against the domain models described in earlier sections.

ACKNOWLEDGMENTS

Parts of this work have been funded through the Sense-T data platform project. Sense-T is a partnership program between the University of Tasmania, the Tasmanian Government, and CSIRO. It is also funded by the Australian Government.

REFERENCES

Akyildiz, Ian F, and Ismail H Kasimoglu. (2004). "Wireless Sensor and Actor Networks: Research Challenges." *Ad Hoc Networks* 2 (4): 351–367. doi:http://dx.doi.org/10.1016/j.adhoc.2004.04.003.

Botts, M, G Percivall, C Reed, and J Davidson. (2008). "OGC Sensor Web Enablement: Overview and High Level Architecture." In *GeoSensor Networks*, 4540:175–190. Springer. http://www.springerlink.com/index/10.1007/978-3-540-79996-2.

Bröring, Arne, Johannes Echterhoff, Simon Jirka, Ingo Simonis, Thomas Everding, Christoph Stasch, Steve Liang, and Rob Lemmens. (2011). "New Generation Sensor Web Enablement." *Sensors* 11 (3): 2652–2699.

Cox, S. (2010). "Geographic Information: Observations and Measurements–OGC Abstract Specification Topic 20." *URL: Http://Portal. Opengeospatial. Org/Files*.

Cox, Simon JD, Jonathan Yu, and Terry Rankine. (2014). "SISSVoc: A Linked Data API for Access to SKOS Vocabularies." *Semantic Web* 7 (1): 9–24.

CSIRO. (2016). "The Big Six: CSIRO's Plans for Our Future." September 20. https://www.csiro.au/en/News/News-releases/2016/The-big-six-CSIROs-plans-for-our-future.

Delin, Kevin A., and Shannon P. Jackson. (2001). "The Sensor Web: A New Instrument Concept." In *Proc. SPIE*. Vol. 4282. http://reviews.spiedigitallibrary.org/pdfaccess.ashx?url=/data/conferences/spiep/34351/1_1.pdf.

DSITI. (2017). "SILO Climate Data." July. https://www.longpaddock.qld.gov.au/silo/.

Fowler, Martin. (2016). "Serverless Architectures." *Martinfowler.Com*. August. https://martinfowler.com/articles/serverless.html.

Hart, Jane K., and Kirk Martinez. (2006). "Environmental Sensor Networks: A Revolution in the Earth System Science?" *Earth-Science Reviews* 78 (3): 177–91. doi:10.1016/j.earscirev.2006.05.001.

Hugo, Daniel, Ben Howell, Claire D'Este, Greg Timms, Chris Sharman, Paulo De Souza, and Simon Allen. (2011). "Low-Cost Marine Monitoring: From Sensors to Information Delivery." In *OCEANS 2011*, 1–7. IEEE. http://ieeexplore.ieee.org/abstract/document/6106957/.

Innes, Trevor Craig. (1996). "Improving Seasoned Hardwood Timber Quality : With Particular Reference to Collapse." Phd, University of Tasmania. http://eprints.utas.edu.au/20796/.

John Caron, UCAR, and E. Davis. (2006). "UNIDATA's THREDDS Data Server." In *22nd International Conference on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology*. https://ams.confex.com/ams/Annual2006/techprogram/paper_104590.htm.

Johnson, I. R., D. F. Chapman, V. O. Snow, R. J. Eckard, A. J. Parsons, M. G. Lambert, and B. R. Cullen. (2008). "DairyMod and EcoMod: Biophysical Pasture-Simulation Models for Australia and New Zealand." *Australian Journal of Experimental Agriculture* 48 (5): 621–631.

Kerry Taylor & Arun Ayyagari. (2006). *Research Topics in Semantic Sensor Networks: Preface to the Proceedings of the Semantic Sensor Network Workshop 2006*.

M.T. Harrison, R.P. Rawnsley, D. Henry, C. Peters, and C. Sharman. (2017). "From Data to Decisions: Using Real-Time Climate Forecasts in Biophysical Models in a Cloud-Based Online Platform for Agricultural Decision-Makers (to Appear)." In .

Sheth, A, C Henson, and S S Sahoo. (2008). "Semantic Sensor Web." *IEEE Internet Computing* 12 (4): 78–83. doi:10.1109/MIC.2008.87.

Taylor, Peter. (2012). "OGC WaterML 2.0: Part 1-Timeseries." *Open Geospatial Consortium*.

Terhorst, Andrew, Peter Taylor, Brad Lee, Chris Peters, Christian Malewski, and Delft OpenWater. (2011). "Enabling Near Real-Time Water Resource Management Via The Sensor Web." *OpenWater Symposium*, 49.

University of Tasmania. (2017a). "Forest and Wood." *Forest and Wood*. http://forestandwood.sense-t.org.au/.

———. 2017b. "Home - Sense-T - from Sensing to Intelligence." http://www.sense-t.org.au/.

Workflow Patterns Initiative. (2017). "Workflow Patterns Home Page." http://www.workflowpatterns.com/.

Wu, C., R. Tobar, K. Vinsen, A. Wicenec, D. Pallot, B. Lao, R. Wang, et al. (2017). "DALiuGE: A Graph Execution Framework for Harnessing the Astronomical Data Deluge." *Astronomy and Computing* 20 (July): 1–15. doi:10.1016/j.ascom.2017.03.007.