Semantic Information Integration with Transformations for Stream Reasoning

Fredrik Heintz
Department of Computer and Information Science
Linköping University, Sweden
fredrik.heintz@liu.se

Daniel de Leng
Department of Information and Computing Sciences
Utrecht University, The Netherlands
d.n.deleng@students.uu.nl

***Abstract*—The automatic, on-demand, integration of information from multiple diverse sources outside the control of the application itself is central to many fusion applications. An important problem is to handle situations when the requested information is not directly available but has to be generated or adapted through transformations. This paper extends the semantic information integration approach used in the stream-based knowledge processing middleware DyKnow with support for finding and automatically applying transformations. Two types of transformations are considered. Automatic transformation between different units of measurements and between streams of different types. DyKnow achieves semantic integration by creating a common ontology, specifying the semantic content of streams relative to the ontology and using semantic matching to find relevant streams. By using semantic mappings between ontologies it is also possible to do semantic matching over multiple ontologies. The complete stream reasoning approach is integrated in the Robot Operating System (ROS) and used in collaborative unmanned aircraft systems missions.**[1]

I. Introduction

The information available to modern systems such as robots is often incremental in nature. A flow of incrementally available time-stamped information is called a *stream*. As the number of sensors and other sources of streams increases there is a growing need for incremental reasoning over streams to draw relevant conclusions and react to new situations with minimal delays. We call such reasoning *stream reasoning*. Reasoning over incrementally available information is needed to support important functionalities such as situation awareness, execution monitoring, and planning [1].

One major issue with regards to such stream reasoning is its integration in robotic systems. To do symbolic reasoning it is necessary to map symbols to streams in a robotic system, which provide them with the intended meaning for the particular robot. This is often done syntactically by mapping symbols to streams based on their names. This makes a system fragile as any changes in existing streams or additions of new streams require that the mappings be checked and potentially changed. This also makes it hard to reason over streams of information from multiple heterogeneous sources, since the name and content of streams must be known in advance.

[1]This work is partially supported by grants from the Swedish Foundation for Strategic Research (SSF) project CUAS, the Swedish Research Council (VR) Linnaeus Center CADICS, and the Center for Industrial Information Technology CENIIT.

To address these problems we have developed a semantic matching approach using semantic web technologies [2]. An ontology is used to define a common vocabulary over which symbolic reasoning can be done. Streams are annotated with ontological concepts to make semantic matching between symbols and streams possible. The output of the semantic matching is a stream specification which is used to produce a stream of time states corresponding to a temporal model of the symbols. One major advantage is that the semantics of a stream can now be described by the creator of the stream and then found by anyone based on this semantic annotation. Previously the user had to know the meaning of the content of streams to select the appropriate ones.

The semantic information integration approach extends the stream reasoning functionality of the stream-based knowledge processing middleware framework DyKnow [3], [4]. DyKnow is integrated in the Robot Operating System (ROS) [5], which makes it available for a wide variety of robotic systems.

An important problem for automatic, on-demand, integration of information from multiple diverse sources outside the control of an application itself is to handle situations when the requested information is not directly available but has to be generated or adapted through transformations. The work in this paper further extends the semantic information integration approach with support for finding and automatically applying transformations. Two types of transformations are considered. Automatic transformation between different units of measurements and transformations between different types of streams.

II. Stream Reasoning

One technique for incremental reasoning over streams is progression of metric temporal logic formulas. This provides real-time incremental evaluation of logical formulas as new information becomes available. First order logic is a powerful technique for expressing complex relationships between objects. Metric temporal logics extend first order logics with temporal operators that allow metric temporal relationships to be expressed. For example, our temporal logic, which is a fragment of the Temporal Action Logic (TAL) [6], supports expressions which state that a formula F should hold within 30 seconds and that a formula F' should hold in every state between 10 and 20 seconds from now. This is similar to the well known Metric Temporal Logic [7]. Informally, $\Diamond_{[\tau_1,\tau_2]}\,\phi$ ("eventually") holds at τ iff ϕ holds at

some $\tau' \in [\tau + \tau_1, \tau + \tau_2]$, while $\Box_{[\tau_1,\tau_2]}\, \phi$ ("always") holds at τ iff ϕ holds at all $\tau' \in [\tau + \tau_1, \tau + \tau_2]$. Finally, $\phi\, \mathsf{U}_{[\tau_1,\tau_2]}\, \psi$ ("until") holds at τ iff ψ holds at some $\tau' \in [\tau + \tau_1, \tau + \tau_2]$ such that ϕ holds in all states in (τ, τ').

We have for example used this expressive metric temporal logic to monitor the execution of complex plans [8] and to express conditions for when to hypothesize the existence and classification of observed objects in an anchoring framework [9]. For example, suppose that a UAV supports a maximum continuous power usage of M, but can exceed this by a factor of f for up to τ units of time, if this is followed by normal power usage for a period of length at least τ'. The following formula can be used during execution to detect violations of this specification: $\Box \forall \mathit{uav} : (\mathsf{power}(\mathit{uav}) > M \rightarrow \mathsf{power}(\mathit{uav}) < f \cdot M\, \mathsf{U}_{[0,\tau]}\, \Box_{[0,\tau']}\, \mathsf{power}(\mathit{uav}) \leq M)$.

The semantics of these formulas are defined over infinite state sequences. To make metric temporal logic suitable for stream reasoning, formulas are incrementally evaluated using progression over a stream of time-stamped states. The result of progressing a formula through the first state in a stream is a new formula that holds in the remainder of the state stream if and only if the original formula holds in the complete state stream. If progression returns true (false), the entire formula must be true (false), regardless of future states. Even though the size of a progressed formula may grow exponentially in the worst case, it is always possible to use bounded intervals to limit the growth. It is also possible to introduce simplifications which limits the growth for many common formulas [10].

III. Semantic Information Integration

A temporal logic formula consists of symbols representing variables, sorts, objects, features, and predicates besides the symbols which are part of the logic. A *feature* represents a property or relation that may change values over time. A *sort* is a collection of objects. Consider $\forall x \in \mathsf{UAV} : x \neq \mathsf{uav1} \rightarrow \Box\, \mathsf{XYDist}[x, \mathsf{uav1}] > 10$, which has the intended meaning that all UAVs, except uav1, should always be more than 10 meters away from uav1. This formula contains the variable x, the sort UAV, the object uav1, the feature XYDist, the predicates $\neq$ and $>$, and the constant value 10, besides the logical symbols. To evaluate such a formula an interpretation of its symbols must be given. Normally, their meanings are predefined. However, in the case of stream reasoning the meaning of features can not be predefined since information about them becomes incrementally available. Instead their meaning has to be determined at run-time. To evaluate the truth value of a formula it is therefore necessary to map feature symbols to streams, synchronize these streams, and extract a timed state sequence where each state assigns a value to each feature for a particular time-point [10].

In a system consisting of streams, a natural approach is to syntactically map each feature to a single stream. This works well when there is a stream for each feature and the person writing the formula is aware of the meaning of each stream in the system. However, as systems become more complex and if the set of streams or their meaning changes over time it is much harder for a designer to explicitly state and maintain this mapping. Therefore automatic support for mapping features in a formula to streams in a system is needed.

The purpose of this matching is to find one or more streams for each feature whose content matches the intended meaning of the feature. This is a form of semantic matching between features and contents of streams. The process of matching features to streams in a system requires that the meaning of the content of the streams is represented and that this representation can be used for matching the intended meaning of features with the actual content of streams.

The same approach can be used for symbols referring to objects and sorts. It is important to note that the semantics of the logic requires the set of objects to be fixed. This means that the meaning of an object or a sort must be determined for a formula before it is evaluated and then stay the same. It is still possible to have different instances of the same formula with different interpretations of the sorts and objects.

Our goal is to automate the process of matching the intended meaning of features, objects, and sorts to content of streams in a system. Therefore the representation of the semantics of streams needs to be machine readable. This allows the system to reason about which stream content corresponds to which symbol in a logical formula. The knowledge about the meaning of the content of streams needs to be specified by a user, even though it could be possible to automatically determine this in the future. By assigning meaning to stream content the streams do not have to use predetermined names, hard-coded in the system. This also makes the system domain independent meaning that it could be used to solve different problems in a variety of domains without reprogramming.

In a previous paper [2] we presented a solution based on creating an ontology acting as a common vocabulary of features, objects, and sorts, a language (SSL_T) for representing the content of streams relative to the ontology, and a semantic matching algorithm for finding all streams which contain information relevant for a feature, object, or sort in the ontology.

This is in line with recent work on semantic modeling of sensors [11], [12] and on semantic annotation of observations for the Semantic Sensor Web [13]–[15]. An interested approach is a publish/subscribe model for a sensor network based on semantic matching [13]. The matching is done by creating an ontology for each sensor based on its characteristics and an ontology for the requested service. If the sensor and service ontologies align, then the sensor provides relevant data for the service. This is a complex approach which requires significant semantic modeling and reasoning to match sensors to services. Our approach is more direct and avoids most of the overhead.

The presented approach also bears some similarity to work by Whitehouse et al. [16] as both use stream-based reasoning and are inspired by semantic web services. One major difference is that we represent the domain using an ontology while they use a logic-based markup language that supports 'is-a' statements. Additionally, they present a declarative inference composition engine that can be used to filter events, whereas DyKnow concerns itself with the

evaluation of spatio-temporal formulas. Currently, unlike Whitehouse et al., we do not yet support Quality of Service annotations. This remains an open challenge.

Our previous solution handles cases when there is a direct semantic match between the ontological concept and the semantic stream annotation. This means that the requested information must already be available in the desired form. However, most systems do not contain all potential streams but only those that are actually needed. In many cases, a system would be able to produce a matching stream. However, the creation of streams is separated from the finding of streams. In the approach described in this paper, transformations between streams are also annotated and made available to the semantic matching process. This makes it possible to either find existing streams or generate a matching stream using the available transformations.

In this paper we describe two extensions to our previous approach. The first extension is to explicitly represent and reason about units of measurement, which is very important for applications using data from many different organizations or sensors. The second extension handles transformations between different features, such as transforming the barometric pressure to an estimation of the altitude of a UAV, by annotating transformations and including these in the semantic matching process.

IV. Supporting Units of Measurement

In this section, we introduce *units of measurement* into the DyKnow framework. Units of measurement are used to describe the magnitude of a *physical quantity*. A physical quantity is something that can be quantified through measurement, e.g. length or mass. A unit of measurement determines a scale, such as 'meter' or 'Pascal', in a dimension, where the dimension is a combination of relevant physical quantities. Since DyKnow is designed to create models of the physical world and reason about the world using these models, support for units of measurement is important. Suppose for example that we are interested in monitoring that the altitude of every known UAV is greater than 10 feet:

$$\forall x \in \mathsf{UAV} : \mathsf{Altitude}[x] \geq 10\text{ft}. \tag{1}$$

This formula checks whether every object in the domain UAV has the feature Altitude whose current numerical value is at least 10 feet. Another example formula is

$$\Diamond_{[0,1000]}\Box_{[0,30]}\mathsf{XYDist}[\mathsf{uav1}, \mathsf{target}] < 2\text{m}, \tag{2}$$

which checks whether there are, between now and 1000 time units from now, at least 30 consecutive time units where the distance in the xy-plane between objects uav1 and target is less than two meters.

To evaluate these formulas, a number of issues have to be solved. The first issue is how to handle explicit units of measurement in stream reasoning formulas. Units such as feet in Formula 1 and meters in Formula 2 are represented by the labels 'ft' and 'm' respectively. To handle these formulas, the temporal logical language has to be extended with units of measurements and the semantics have to be extended to give the right interpretation. The second issue is how to support interoperability when different streams use different units of measurement and when a formula uses different units compared to the streams. In the examples, both metric and imperial units of measurement are used. Furthermore, a feature such as Altitude can naturally be described in terms of both meters and feet, even in the same application. The third issue is to handle formulas which refer to physical quantities without explicit units of measurement, which should be treated differently compared to non-physical quantities such as the number of UAVs. In Formula 2, for example, the time units are not explicitly specified, even though they are physical quantities associated with time.

To handles these issues, we propose a solution which extends our ontology by representing units of measurements and their relations, extends our semantic annotation of streams to also include the unit of measurement, and extends our semantic matching functionality by taking the units of measurements into consideration. The rest of this section describes the details.

A. Ontology Support for Units of Measurements

In order to support formulas containing units of measurement, a representation of these units is needed. One approach is to construct a general representation that maps labels to units and utilizes a conversion table between different units. This is done for example with the Unified Code for Units of Measures (UCUM) proposed by Schadow et al. [17]. Additionally, UCUM makes use of prefixes such as 'kilo' to reduce the number of explicit conversions required. However, one downside of this approach is its complexity due to generality, which may be higher than necessary for many application domains. Furthermore, it has a fixed custom system of units described by length, time, mass, charge, temperature, luminous intensity, and angle. Depending on the application, this system of units may be considered impractical.

An alternative to the use of a conversion table is to use an ontology representing units of measurement, which may also model conversion tables such as UCUM. One example of this is the Measurement Units Ontology (MUO) [18]. Rijgersberg et al. [19] analyze existing ontologies that incorporate units of measurement, including the Suggested Upper Merged Ontology (SUMO) [20], a subset of the Semantic Web for Earth and Environmental Terminology (SWEET) ontology [21], the EngMath ontology [22], ScadaOnWeb [23], and the OpenMath units and dimension CD groups [24]. The analysis for example looked at concepts such as unit prefixes and resulted in the Ontology of Units of Measure and Related Concepts (OM) [25].

Clearly, there exist many variations of units of measurement representations, which is an ongoing area of research. Instead of selecting one approach, DyKnow supports all ontologies that satisfies a small number of requirements with regards to the properties of the units of measurement ontology. These requirements are outlined below.

a) Units of measurement are related to their physical quantities: It is possible to describe many physical quantities

by a signature $\sigma = (\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta)$ corresponding to the unit $L^\alpha M^\beta T^\gamma I^\delta \Theta^\epsilon N^\zeta J^\eta$, where L represents length, M represents mass, T represents time, I represents electric current, Θ represents thermodynamic temperature, N represents amount of substance, and J represents luminous intensity [19]. For example, $\sigma = (1, 0, -1, 0, 0, 0, 0)$ describes speed as length (distance) divided by time. By associating (with an `unit-of` relation in OWL) for example {'meter', 'foot'} to length L and {'second'} to time T, speed L/T can be described in two different ways, either as meters per second or feet per second. While we have chosen to use this seven-dimensional system in this paper, it must be pointed out that any set of physical quantities may be used. This includes physical quantities unmentioned, e.g. angle as used in UCUM. The choice of base physical quantities, described by a signature of equal dimension, determines which physical quantities may be derived. In this paper, only length, mass, and time are used, which means for the purpose of this paper a signature $\sigma' = (\alpha, \beta, \gamma)$ would have sufficed. However, for completeness, the seven-dimensional system is used.

b) Every physical quantity must have a default representative unit of measurement: The International System of Units (SI) uses meter, kilogram, second, ampere, kelvin, mole, and candela. However it co-exists alongside various other systems of units such as the imperial system. Additionally, minor variations such as using the gram as opposed to the kilogram for M may sometimes be desired. However, for every physical quantity there must be one default representative unit of measurement. These units of measurement make up the *system of units*, and are modeled through a relation in the ontology called `base` from a physical quantity to a unit of measurement.

c) Every physical quantity is closed under unit conversion: This means that every unit of measurement is guaranteed to have a conversion function f and its inverse f^{-1} to the representative unit of a physical quantity, which makes it possible to convert any unit within a physical quantity to any other unit within the same physical quantity (see Theorem 1 in Section IV-C). For the representative unit of measurement, f and f^{-1} are identity functions. In the case of units on the ratio scale, the conversion function f will be of the form $f(x) = v \times x$, where v represents a conversion factor. For example, the conversion function for 'feet' to 'meters' is defined as $f(x) = 0.3048 \times x$. This conversion factor is modeled in the ontology as part of the unit of measurement's corresponding concept.

With a unit ontology satisfying the above criteria, the name of the concept of each unit of measurement can be used directly in stream reasoning formulas. By using equality relations, it is possible to represent that for example 'meter', 'metre' and 'm' are equivalent concepts that can be used in formulas interchangeably. Furthermore, the conversion factor between two commensurable units of measurement is represented in the ontology as well.

To give every feature a default unit of measurement, the feature ontology used by DyKnow is extended with a `dimension` relation that associates a combination of physical quantities to a feature. This is used to resolve situations where there is no explicit unit of measurement and when the same feature is represented in different ontologies, where a common unit can not be assumed. To handle non-physical quantities, e.g. the number of known UAVs, a special unit (`no_unit`) representing this is added. This is also implicitly modeled in the ontology through the absence of a `dimension` relation.

Derived units of measurement are units of measurement that are composed of other units of measurement, and have their own label. It can be useful to represent these aggregate units of measurement in the ontology. As an example, take Pascal (Pa), which is defined as $1\mathrm{Pa} = 1\mathrm{kg}/(\mathrm{m} \cdot \mathrm{s}^2)$, or alternatively `[kg.m-1.s-2]` with signature $\sigma = (-1, 1, -2, 0, 0, 0, 0)$. Note that kilogram itself is a concept and not derived from gram; while it is possible to automatically derive some units of measurements using prefixes [17], this is beyond the scope of this paper. In order to refer to Pascal directly, it is necessary to include it in the ontology. However, it may be undesirable to introduce *pressure* as a base physical quantity into the system of units. Therefore, it needs to be related to its base concepts 'kilogram', 'meter', and 'second' alongside their respective powers, which constitutes a ternary relation. Because OWL only supports binary relations, we choose to specify six binary derivation relations `derived`; one for every power in the set $\{-3, -2, -1, +1, +2, +3\}$. The choice to limit the relations to this set is based on the fact that most units of measurement seem to be contained and if necessary it can be changed. Note that the zero-power relation is always omitted as it is indirectly represented by the absence of a `derived` relation.

Pressure can now be represented using `derived` relations to length, mass, and time. By including *pressure* as a non-base physical unit, a new physical quantity with its own default unit of measurement is introduced. Assuming 'kilogram', 'meter', and 'second' are the default units for mass, length, and time respectively, it is possible to represent 'pascal' as the default unit for pressure with the signature $\sigma = (-1, 1, -2, 0, 0, 0, 0)$.

However, representing 'bar' as the default unit for pressure, which is defined as 10^5 Pa, is more difficult. In order to represent 'bar', an additional conversion factor called the *derived conversion factor* is required. This conversion factor is based on the default units of the derived physical quantity's base physical quantities, which in the case of 'bar' is 10^5. This value is modeled as part of the physical quantity associated with 'bar', being pressure. An example unit ontology modeling pressure is shown in Figure 1, with units of measurement in a light shade and physical quantities in a darker shade. Relations with open arrow heads represent `unit-of` relations. By incorporating `derived` relations in the ontology, a transformation chain is created from these derived units of measurement to their base units of measurement. This makes it possible to semantically annotate values in streams with derived units of measurement.

One strength of the proposed requirements is that existing ontologies may be used with a few changes. The main requirement is that physical quantities and units of measurement must be identified as such in the ontology, for

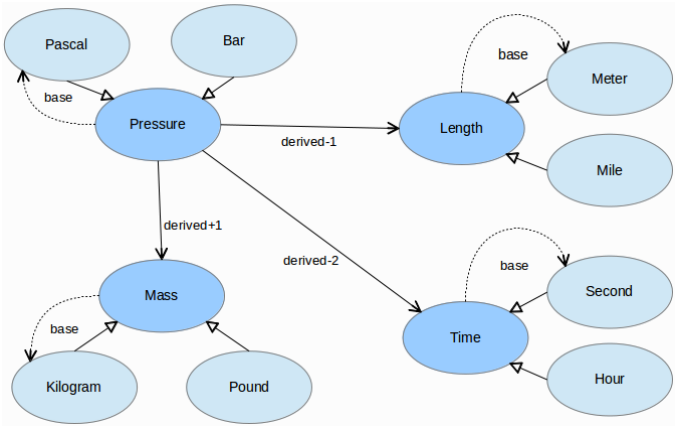

Fig. 1: Example unit ontology with Physical Quantity and Unit of Measurement concepts omitted for clarity.

example through a concept with the same name. Derived physical quantities can then be identified by adding `derived` relations with other physical quantities, where base physical quantities are implicit through the absence of `derived` relations originating from those physical quantities. Upper level ontologies such as SWEET and SUMO already contain concepts for (physical) quantities, which can be extended.

B. Semantic Annotation of Streams with Unit of Measurements

To support semantic information integration, streams in DyKnow are annotated with their semantic content, normally one or more features. With the introduction of units of measurement, streams containing features of physical quantities are also annotated with their units of measurement. This annotation allows DyKnow to reason about the unit of measurement of a stream. This is important both for the semantic matching and for preventing mix-ups between different measurement systems, with potentially catastrophic results.

Since DyKnow is realized using ROS, streams correspond to ROS *topics*. To support units of measurement, the previously proposed Semantic Specification Language for Topics (SSL_T) [2] is extended. The extended version of SSL_T, inspired by the *unit expressions grammar* [17], is presented in Listing 1. The change compared to the old version is that topic specifications now contain a unit of measurement (`unit_list`). Note that the extended grammar allows for aggregated units of measurement, making it possible to represent e.g. acceleration ($\mathrm{m/s^2}$) as `[m.s-2]`, where 'm' and 's' are concepts in the ontology related to length and time respectively. From this notation, the signature $\sigma = (1, 0, -2, 0, 0, 0, 0)$ can be inferred, which is necessary for unit conversion. Non-physical quantities, e.g. the number of known UAVs, have `no_unit` as their default unit of measurement. If no explicit unit is given, then the unit is assumed to be `no_unit`.

Listing 1: Extended formal grammar for SSL_T.

```
spec          : expression+ ;
expression    : 'topic' topic_name 'contains'
                feature_list ;
topic_name    : NAME ':' NAME ;
feature_list  : feature (',' feature)* ;
feature       : feature_name '=' MSGFIELD
                unit_list? for_part? ;
feature_name  : NAME '(' feature_args ')' ;
feature_args  : feature_arg (',' feature_arg)* ;
feature_arg   : entity_name alias? ;
for_part      : 'for' entity (',' entity)* ;
entity        : sort | object ;
entity_name   : NAME;
unit_list     : '['? unit ('.' unit)* ']'? ;
unit          : NAME power? | 'no_unit';
power         : '-'? NUMBER ;
alias         : 'as' NAME ;
object        : entity_full ;
sort          : sort_type entity_full ;
entity_full   : NAME '=' MSGFIELD ;
sort_type     : 'some' | 'every' ;
NAME          : ('a'..'z'|'A'..'Z'|'0'..'9')+ ;
NUMBER        : ('0'..'9')+ ;
MSGFIELD      : NAME '.' NAME ;
```

Listing 2: Example SSL_T specifications.

```
topic topic1:UAVMsg contains Altitude(UAV)=msg.alt
    ft for every UAV=msg.id
topic topic2:DistMsg contains XYDist(uav1, uav2)=msg
    .dist for uav1=msg.id1, uav2=msg.id2
topic topic3:UAVMsg contains Acceleration(uav3)=msg.
    acc [mi.h-2] for uav3=msg.id
```

As an example, Listing 2 contains topics relevant for Formulas 1 and 2. topic1 has the message type UAVMsg, and contains the feature Altitude for every object of sort UAV. Additionally, it states that Altitude uses the unit feet (ft), which means that the numerical values of field msg.alt describe the feature Altitude in feet. For topic2 the message type is DistMsg. This topic contains feature XYDist for the objects uav1 and uav2. Unlike the case with topic1, no explicit unit of measurement is given. In such a case, DyKnow either uses the default unit of measurement for feature XYDist as represented in the ontology, or returns a warning, depending on the configuration. If, for example, the default unit is meters then the numerical value of field msg.dist is assumed to have the unit meters. Topic topic3 contains messages of type UAVMsg containing the Acceleration feature for object uav3. The unit of measurement for Acceleration is described as miles per hour squared ($\mathrm{mi/h^2}$), which is a derived unit.

C. Semantic Matching with Units of Measurement

The semantic matching problem is as follows. Given an ontology, a stream specification, and a parameterized feature find a set of streams which allows the estimation of the value of a feature over time. With the introduction of units of measurement, the previous semantic matching algorithm [2] must be extended to take into account these units of measurement during matching. The new algorithm is shown in Procedure 1.

An important aspect of the new functionality is that it can be treated as an optional extension. Previously, the matching procedure finished after finding matching features. With the addition of units of measurement, an additional test is done to make sure that the units of measurement are aligned or can be converted so that they are aligned.

Procedure 1 Semantic matching with units of measurement

Input: A well-formed formula Φ and a set of stream specifications S
Output: Set of matching stream specifications S'

```
Set S' ← ∅
List F ← ExtractGroundFeatures(Φ)
for all f ∈ F do                                  ▷ Match and convert
    for all s ∈ MatchSpecs(S,f) do
        if Unit(s) = Unit(f) then
            S' ← S' ∪ {s}
        else if Convertable(s, Unit(f)) then
            S' ← S' ∪ {Convert(s, Unit(f))}
        end if
    end for
end for
return S'
```

In order to illustrate *unit alignment* in semantic matching, consider a slightly altered Formula 1: Altitude[uav3] $\geq$ 10ft. In this formula, we have sort UAV, feature Altitude, and unit of measurement 'ft' for feet. Running the algorithm on the formula results in Altitude[uav3] being extracted and inserted into a list. Every element of this list is then checked to see if it is grounded. If the extracted feature had for example been Altitude[UAV], then it would have been grounded for every object of the sort UAV yielding a new (expanded) list {Altitude[uav1], Altitude[uav2], Altitude[uav3]}. For Altitude[uav3], no expansion is necessary (or even possible).

Next, the topic specifications are matched. In our example, the three topic specifications from Listing 2. As per the non-extended semantic matching approach, topics containing information for feature Altitude[uav3] are selected. Only topic topic1 contains the Altitude feature for the sort UAV, which object uav3 is part of. Therefore, topic1 is selected.

Finally, the unit of measurement for feature Altitude is checked. If the default unit of measurement for Altitude is 'ft', the matching procedure is done and returns topic topic1. However, if the default unit is different from 'ft', this qualifies as a *misalignment problem*, and a unit conversion mechanism is used to fix the alignment if possible. When this happens, the resulting realigned topic is returned as a match.

The process of *unit alignment* is defined here as the process of converting the unit of measurement of a stream containing a feature in order to match the desired unit of measurement. Recall that every feature described by a physical unit can be described by a signature σ. For example, 'area' is described by $\sigma = (2, 0, 0, 0, 0, 0, 0)$, without specifying the unit of measurement used, which could be e.g. meters, feet, or furlong.

Theorem 1 (Commensurability): Given two physical quantities on the ratio scale, described by signatures σ_1 and σ_2, then if $\sigma_1 = \sigma_2$, there exists a transformation function f that converts the unit of measurement described by σ_1 to the unit of measurement described by σ_2.

Proof: Assume the criteria specified in Section IV-A hold. Given two physical quantities, described by signatures σ_1 and σ_2, such that $\sigma_1 = \sigma_2$, we denote u_1 and u_2 to be the units of measurement for σ_1 and σ_2 respectively. From the closure criterion (c), there exists a function $f_1(u_1) = u_d$ and a function $f_2(u_2) = u_d$, where u_d is the default unit. Because of the requirement, a conversion function exists as well as its inverse, i.e. $f_2^{-1}(u_d) = u_2$. This means that $f = f_1 \circ f_2^{-1}$, and therefore $\exists f : f = f_1 \circ \ldots \circ f_n$. It is thus shown that there exists a transformation function f that converts the unit of measurement described by σ_1 to the unit of measurement described by σ_2. ■

Recall that physical quantities can operate at different scales. For example, units of measurement such as meters, seconds, and Kelvin all operate on the ratio scale. It is also possible to apply sequences of transformation on derived units. As an example, take the conversion of `[mi.h-1]` to `[m.s-1]`, both of which share the same signature; $\sigma = (1, 0, -1, 0, 0, 0, 0)$. It is guaranteed that a transformation sequence f_1 and f_2 converting from miles to meters and from hours to seconds respectively exists. Given some value x miles per hour, $f_1 \circ f_2^{-1}$ can be applied to x to yield the speed in meters per second, which matches the signature σ. While this is possible for units on the ratio scale, it is more challenging for units such as °C or °F, which operate on the interval scale. One possibility to model these conversion functions is by using the built in math functions in the Semantic Web Rule Language (SWRL) [26], but this is left for future work.

V. Supporting Feature Transformations

So far, our semantic information integration approach only considers already existing streams. The previous section introduced a solution to handle the case when there exist streams producing the requested feature, but where none of those streams have a matching unit of measurement. This section describes an approach to automatically create a matching stream by transforming existing streams.

In DyKnow, *computational units* are used to transform streams. A computational unit takes one or more streams as input and generates one more new streams as output. For example, a computational units may act as a refinement process by taking a stream of time-stamped vehicle positions and generating a stream of speed estimations for the tracked vehicles. From a knowledge representation point of view, this computational unit transforms a position feature into a speed feature.

To support the automatic and on-demand creation of streams using feature transformations we introduce a language for semantic annotations of computational units together with a new semantic matching algorithm. The semantic annotation language is used to annotate the inputs and outputs of computational units with their features and units of measurement. This kind of semantic annotation of inputs and outputs is in line with service composition approaches [27], [28]. Then a new semantic matching algorithm uses the semantic annotations of streams and transformations to either find or create a matching stream. If it is necessary to create a stream, it is created by transforming one or more existing streams using one or more computational units.

A. Semantic Annotation of Feature Transformations

The purpose of the semantic annotation of transformations is to make it possible to automatically find a transformation from one feature to another. If a single transformation is not enough, a chain of transformations might be needed. Another benefit of semantic annotations of transformations is that it provides strong semantic typing of computational units.

In DyKnow a computational unit is seen as a function from a set of streams (the inputs) to another set of streams (the outputs). For the purpose of semantic matching we will consider a computational unit C as a function from a set of features to a single feature, $C : \mathcal{F}^n \rightarrow \mathcal{F}$, where $\mathcal{F}$ denotes the set of all possible features in the ontology. The reason is that the ontology represents features, not streams, and since streams are annotated with features the connection is clear. In order to also represent the unit of measurement of a feature, each feature is represented as a tuple $\langle name, unit \rangle$. The feature name must refer to a feature in the feature ontology.

For example, imagine a stream s annotated with the feature f and two computational units C_1 and C_2. The transformations applied by these computational units are represented by functions $t_1 : f_{11} \rightarrow f_{12}$ and $t_2 : f_{21} \rightarrow f_{22}$. If it is the case that $f = (\mathsf{Altitude}, \mathrm{m})$ and $f_{11} = (\mathsf{Altitude}, \mathrm{m})$, then it follows that stream s and computational unit C_1 match since the same feature is associated with stream s and the input of C_1, i.e. $f = f_{11}$. However, if it is the case that $f_{21} = (\mathsf{Altitude}, \mathrm{ft})$, then C_2 does not match the stream s unless there is a unit transformation between m and ft.

Recall that in Theorem 1 the notion of commensurability was introduced in terms of signatures σ. We now introduce a function $\sigma : \mathcal{F} \rightarrow \Sigma$, where Σ is the set of all possible signatures σ. Concretely, this means that the function $\sigma(f)$ for a feature f yields the signature of the feature. In the previous example, this means that $\sigma(f) = \sigma(f_{11}) = \sigma(f_{21}) = (1, 0, 0, 0, 0, 0, 0)$. Because the signatures are the same, it can be concluded that stream s is commensurable with the inputs of computational units C_1 and C_2. In cases of misalignment, it is possible to distinguish between *commensurable misalignment*, which can be realigned through unit conversion, and *strict misalignment*, which can not be realigned through unit conversion. This distinction makes it possible to automatically introduce required unit conversions.

Listing 3: Formal grammar for SSL_{TF}.

```
spec          : expression+ ;
expression    : 'transform'
                'from' feature_list 'to' feature
feature_list  : feature (',' feature)* ;
feature       : NAME unit_list ;
unit_list     : '['? unit ('.' unit)* ']'? ;
unit          : NAME power? | 'no_unit';
power         : '-'? NUMBER ;
NAME          : ('a'..'z'|'A'..'Z'|'0'..'9')+ ;
NUMBER        : ('0'..'9')+ ;
```

To semantically annotate transformations with features and their units of measurement we introduce the Semantic Specification Language for Transformations (SSL_{TF}) as shown in Listing 3. This grammar is based on SSL_T.

Listing 4: Example SSL_{TF} specifications.

```
transform from Distance [m] to Speed [m.s-1]
transform from Distance [km] to Speed [mi.h-1]
transform from NumVehicles no_unit to NumUAVs no_unit
```

As an example, Listing 4 contains a number of transformations between different features. The first transformation takes a feature Distance in (m) to return a feature Speed in $(\mathrm{m/s})$. The second transformation also transforms from distance to speed, but with a different unit of measurement. These transformations rely on the time-stamps in the streams. As a final example, consider the feature NumVehicles which describes the number of known vehicles and has no unit of measurement as it is not related to any physical quantity. Its transformation yields the feature NumUAVs, which describes the number of known UAVs – a subset of vehicles. Such features with no relation to any physical quantity can be modeled with `no_unit`.

B. Semantic Matching with Feature Transformation

The semantic matching algorithm presented in Section IV-C extends the original algorithm by handling the case when there exists a stream with a commensurable signature. In this section we extend the algorithm further by handling the case when there is no commensurable stream but there is a chain of transformations that produces a commensurable stream.

The problem can be stated as follows. Given a desired feature f either find a commensurable stream s or a transformation tree tf generating a commensurable stream. A *transformation tree* is a tree where the interior nodes are computational units and the leaves are streams. The output of a transformation tree is the output of the top-most node (which can be a stream if the tree consists of only a leaf node). A computational unit with n inputs must have exactly n children. A transformation tree describes a *valid transformation* if every leaf node is a stream, each interior node is a computational unit with the right number of children, and the signature of each child is commensurable with the output of its sub-tree.

Procedure 2 describes an algorithm computing a valid transformation tree given a feature using the set of stream specifications S and transformation specifications TF. The algorithm recursively extends the transformation tree one node at the time. To find a matching stream Procedure 1 is used. If no commensurable stream is found, then try to match each input of each computational unit which produces a commensurable stream. As soon as a valid transformation tree is found the algorithm may terminate. To increase the efficiency and avoid cycles previous results are cached. Initially the cache is empty. By adding an empty tree as the default result before matching the feature allows cycles to be detected and handled.

The algorithm is non-deterministic in its choice of where to extend the transformation tree. Many different strategies can be employed to guide the search. If the transformation tree is extended in a depth-first manner and there is a solution it should normally be found quickly. The algorithm can also find optimal solutions by exhaustively trying all possibilities. The optimal solution could for example be the transformation tree with the least number of leaf nodes (streams) or the minimal maximal

Procedure 2 Tree $match$(Feature f)

Assume: A set of stream specifications S, a set of transformation specifications TF, and a cache of results M.
if $f \notin M$ **then**
  $M(f) \leftarrow$ Tree()
  if $\exists s : f' \in S$ such that $\sigma(f) = \sigma(f')$ **then**
    $M(f) \leftarrow$ Tree(s)
  else if $\exists C : f_1, \ldots, f_n \rightarrow f_{n+1} \in TF$ such that $\sigma(f) = \sigma(f_{n+1}) \wedge t_1 = match(f_1) \neq$ Tree() $\wedge \ldots \wedge t_n = match(f_n) \neq$ Tree() **then**
    $M(f) \leftarrow$ Tree(C, $[t_1, \ldots, t_n]$)
  end if
end if
return $M(f)$

length of any path from the root to a leaf. If there are n features and m computational units then the worst case complexity is $O(nm)$ since each feature is only computed at most once and each feature at most tries every computational unit once.

VI. Conclusions and Future Work

We have presented a semantic information integration approach supporting finding and automatically applying stream transformations. Two types of transformations are considered, transformations between different units of measurement and transformations between different features. These extensions make it possible to integrate information that is not directly available but can be generated through transformations. This is an important functionaly as generating desired information on demand is central to many fusion applications.

In order to support transformations between different units of measurement, we presented three criteria that an ontology must adhere to in order to semantically model unit signatures. We have additionally presented an extended formal grammar for SSL_T that allows for the semantic annotation of streams with units of measurement. Since our previous matching algorithm did not consider units of measurement, we have incorporated this as an extension that also takes into account commensurability. Finally, we described how derived units of measurement can be modeled in the ontology, making it possible to provide concepts for derived units such as Pascal.

In order to support feature transformations the inputs and outputs of transformations are semantically annotated. For this, a semantic annotation language for transformations called SSL_{TF}, based on SSL_T, was introduced. Then an algorithm for either finding an existing or creating a new matching stream was introduced. This algorithm leverages the ability to transform between commensurable units of measurement and adds the possibility to chain multiple transformations to generate information that is not directly available.

An open challenge is to support units of measurement that do not operate on the ratio scale, as interval scale units of measurement such as Celsius and Fahrenheit are common. This involves representing more complex conversion functions, e.g. through SWRL and its built in math functions.

Semantic information integration with transformations is essential to stream reasoning about the physical world, since features are often described by physical quantities and information is not always directly available but have to be generated or adapted through transformations.

References

[1] F. Heintz, J. Kvarnström, and P. Doherty, "Stream-based middleware support for autonomous systems," in *Proc. ECAI*, 2010.
[2] F. Heintz and Z. Dragisic, "Semantic information integration for stream reasoning," in *Proc. Fusion*, 2012.
[3] F. Heintz and P. Doherty, "DyKnow federations: Distributing and merging information among UAVs," in *Proc. Fusion*, 2008.
[4] F. Heintz, J. Kvarnström, and P. Doherty, "Bridging the sense-reasoning gap: DyKnow – stream-based middleware for knowledge processing," *J. of Advanced Engineering Informatics*, vol. 24, no. 1, pp. 14–26, 2010.
[5] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.
[6] P. Doherty and J. Kvarnström, "Temporal action logics," in *Handbook of Knowledge Representation*. Elsevier, 2008.
[7] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol. 2, no. 4, pp. 255–299, 1990.
[8] P. Doherty, J. Kvarnström, and F. Heintz, "A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems," *J. of Auton. Agents and Multi-Agent Systems*, vol. 19, 2009.
[9] F. Heintz, J. Kvarnström, and P. Doherty, "Stream-based hierarchical anchoring," *Künstliche Intelligenz*, vol. 27, no. 2, pp. 119–128, 2013.
[10] F. Heintz, "DyKnow: A stream-based knowledge processing middleware framework," Ph.D. dissertation, Linköpings universitet, 2009.
[11] J. Goodwin and D. Russomanno, "Ontology integration within a service-oriented architecture for expert system applications using sensor networks," *Expert Systems*, vol. 26, no. 5, pp. 409–432, 2009.
[12] D. Russomanno, C. Kothari, and O. Thomas, "Building a sensor ontology: A practical approach leveraging iso and ogc models," in *Proc. the Int. Conf. on AI*, 2005.
[13] A. Bröring, P. Maué, K. Janowicz, D. Nüst, and C. Malewski, "Semantically-enabled sensor plug & play for the sensor web," *Sensors*, vol. 11, no. 8, pp. 7568–7605, 2011.
[14] A. Sheth, C. Henson, and S. Sahoo, "Semantic sensor web," *IEEE Internet Computing*, pp. 78–83, 2008.
[15] M. Botts, G. Percivall, C. Reed, and J. Davidson, "OGC® sensor web enablement: Overview and high level architecture," *GeoSensor networks*, pp. 175–190, 2008.
[16] K. Whitehouse, F. Zhao, and J. Liu, "Semantic streams: a framework for composable semantic interpretation of sensor data," in *Proc. EWSN*, 2006.
[17] G. Schadow, C. J. McDonald, J. G. Suico, U. Fhring, and T. Tolxdorff, "Units of measure in clinical information systems," *Journal of the American Medical Informatics Association*, vol. 6, no. 2, 1999.
[18] "MUO." [Online]. Available: http://idi.fundacionctic.org/muo/
[19] H. Rijgersberg, M. Wigham, and J. Top, "How semantics can improve engineering processes: A case of units of measure and quantities," *Advanced Engineering Informatics*, vol. 25, no. 2, 2011.
[20] "SUMO." [Online]. Available: http://www.ontologyportal.org/
[21] "SWEET ontologies." [Online]. Available: http://sweet.jpl.nasa.gov/
[22] T. R. Gruber and G. R. Olsen, "An ontology for engineering mathematics," in *Proc. KR*, 1994.
[23] T. Dreyer, D. Leal, A. Schröder, and M. Schwan, "Scadaonweb - web based supervisory control and data acquisition," in *Proc. ISWC*, 2003.
[24] "OpenMath." [Online]. Available: http://www.openmath.org/ontology/
[25] H. Rijgersberg, M. van Assem, and J. Top, "Ontology of units of measure and related concepts," *Semantic Web*, vol. 4, no. 1, 2013.
[26] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A semantic web rule language combining OWL and RuleML," 2004.
[27] J. Rao and X. Su, "A survey of automated web service composition methods," in *Proc. SWSWPC*, vol. 3387, no. 1, 2005.
[28] S. Dustdar and W. Schreiner, "A survey on web services composition," *International Journal of Web and Grid Services*, vol. 1, no. 1, 2005.