Distributed Contextual Data Fusion with ACIPL

Michael A. McGrath
Department of Computer Science and Engineering
The Ohio State University
Columbus, Ohio 43210-1272, USA
Email: mcgrath.57@osu.edu

Yuan F. Zheng
Department of Electrical and Computer Engineering
The Ohio State University
Columbus, Ohio 43210-1272, USA
Email: zheng@ece.osu.edu

***Abstract*—A system for controlling smart sensor networks is described. The system is called the Adaptive Context Information Processing Language (ACIPL) which will allow explicit use of states of context inferred from sensor readings and algorithmic output for distributed control of data fusion in sensor networks. The detailed description of the language including its use for sensor information separation into raw sensor data, feature, objects, and events by the language is described. Furthermore the concept of context which aids the modeling of sensor data with the data fusion hierarchy is introduced, including its types, properties and applications. ACIPL loosely follows the Joint Directors of Laboratories' Data Fusion Hierarchy and provides a powerful tool to facilitate the use of smart sensor networks for the purpose of sensor data fusion.**

I. Introduction

We propose a system for controlling smart sensor networks called Adaptive Context Information Processing Language (ACIPL). ACIPL integrates tracking and recognition algorithms into distributed entities by loosely following the Joint Directors of Laboratories' Data Fusion Hierarchy (JDL) as presented in [1]. Programs contain execution guards in the form of context statements that relate the entities of the fusion hierarchy in time, space, and user-defined contexts. Users create context daemons that automatically execute on nodes based on sensor hardware capability and presence of other contexts at the nodes. The context daemons continuously adjust their state based on sensor readings and act as a gateway for reasoning about program execution naturally in addition to facilitating data fusion across multiple nodes. Outputs from individual algorithms are in turn represented at the global network level as aggregated contexts. The outputs of algorithms are channeled as contexts that enable the execution of other algorithms and serve as input to algorithms representing entities at the next higher level of the data fusion hierarchy. With this framework it will be possible to develop dynamic programs that build complex representations of the scene while controlling information at local and global levels.

The ACIPL framework borrows concepts from existing sensor network systems. Of the existing sensor network languages, unique characteristics desirable for implementing ACIPL include dynamic reprogramming, region-based programming, portability, and signal processing-friendly data types. Stream and signal-based data types are desirable in ACIPL for supporting DSP algorithms, keeping track of temporal events, and abstracting information flow among established links between different contexts. Region-based coordination of nodes is a key concept for implementing and keeping track of contexts running inside the network. Region-based programming and data streams are implemented in the Regiment/WaveScope Macroprogramming [2] system developed at Harvard and MIT. WaveScope is targeted at low-power raw data collection, whereas ACIPL targets information fusion and inference. Dynamic reprogramming of nodes is a desirable trait among sensor networks. ACIPL's liberal use of sending contexts out into the network to be picked up and run on matching nodes can be seen as dynamic reprogramming, but it is planned that multiple context monitors and data interpretation algorithms will execute on the same node. Multithreading is a desired property of the systems so that nodes may efficiently multitask.

The paper is organized as follows. In Section II, the function of ACIPL which allows lifting of sensor information into data fusion hierarchy of raw sensor data, feature and objects is described. In Section III, the discussion is on the explicit use of context types by ACIPL to aid the modeling of sensor data in the data fusion hierarchy. In Section IV, the use of context in data fusion and in the control the system's state to fuse data across different levels of hierarchy is presented, which is followed by the detailed description of context types, context properties, and ACIPL architecture in Sections V, VI, and VII, respectively. The paper is summarized in Section VIII.

II. Data Fusion Hierarchy

The focus of ACIPL is coordination of tracking applications in sensor networks to allow lifting of sensor information into a representational hierarchy of raw sensor data, features, objects, and events. Organizing entities in the system this way allows for finer separation and flexibility in finding patterns and linking properties of entities across levels in the ACIPL hierarchy, while still facilitating fusion when using algorithms that bypass lower levels of the hierarchy. While the proposed hierarchy may be simplified into sensor data and groups of objects or extended [3] in various ways to the full scope of the JDL fusion hierarchy including impact assessment, the breaking of object and situation assessment into features, objects, and events allows explicit addressing of entity state, identity, and relation to other entities in the system.

ACIPL will initially include spatial and temporal context evaluation. Although entities in upper levels of the hierarchy may be inferred through context rather than directly detected, since they are still based on entities in the real world, they all share the ability to fuse information across the bounds of time and space.

The treatment of contexts as a type allows custom contexts created by programmers to account for relationships among the data that are unique to the intended objects being tracked. Fitting output of each tracking algorithm to a level of the data fusion hierarchy allows relationships to be formed between algorithms that find or create entities at the same level of the data fusion hierarchy; relationships specified by the programmer using context statements allow creation of new entities at a higher level of the data fusion hierarchy.

The creation of a data fusion hierarchy entity and aggregation of its context across a group of nodes are represented as a single program block; the creation of a scene or event relies on chaining together the output of many programs, referring to either generic object contexts or specific algorithms.

Development of raw sensor data into meaningful information is divided into levels that follow the initial stages of the Joint Directors of Laboratories' Data Fusion Hierarchy. Although using context logic to control data fusion is closely related to using multimodal logic solvers to extrapolate events in time for the revised JDL's Level 3 Impact Assessment, ACIPL's focus is on scene understanding through context extraction. In the ACIPL system, the fusion hierarchy is modeled in the order of Raw Sensor Data -> Features -> Objects -> Events.

A. *Fusion Elements*

Fusion entities have unique identification that ties them to their originating algorithm and also to the context of their originating algorithm. Two-part identification is needed so that entities created by the same algorithm running in different parallel contexts may be identified without being fused together or treated as distinct inputs.

Raw Sensor Data is included as a primitive type to allow storage of sensor output for reference by multiple algorithms. Contextual relation of raw sensor data may be based on the range(space), rate (time), and value of the sensor. Fusing data at the raw sensor level may result in an object or a feature; the latter is included to aid in reasoning about attributes of objects over spatial regions and time instead of directly referring to stored sensor output.

Features represent a bridge between concrete raw sensor data and abstract objects and events. With features it is possible to represent patterns in the signal by contextual relation of raw sensor data.

Objects group different types of features together, addressing the signal across a period of time or space elevates the type in the fusion hierarchy.

Coordinating objects in relation to each other creates spatial, temporal, or spatial-temporal events based on the position, value attribute, or position and value attribute changing. Events are considered at the top of the ACIPL fusion hierarchy because it is natural to order object relations across time.

B. *Fusion Scope*

In the sensor network we have both the scope of the individual node and the programs running on it, and the scope of the group created by the context in the network; the ability to address local and global scopes of the same data gives finer control of data fusion. Algorithms may be tailored to run at the node scope, creating small fusion hierarchies at each node and then aggregating them with a group context that correlates the results from many nodes. Or, algorithms can be made to run at the group level, where each level of the hierarchy is decided by the group before being passed on to the next level of the hierarchy. In the latter case, it is important to specify constraints in the context headers so that regions do not overlap.

III. CONTEXT

The main contribution of ACIPL is the explicit use of context types to aid in modeling sensor data with the data fusion hierarchy. Initially ACIPL will focus on modeling and supporting context operators between entities of the same fusion hierarchy level. Doing so affords easier reasoning about contexts on an input-output basis where contexts serve as a way to aggregate data among a program group and feed that into a separate algorithm that produces contexts corresponding to entities at the next higher level of the fusion hierarchy. Because all data has a sense of representing the time window while it was captured and a position that may be inferred, it is natural to include temporal and spatial context as universal properties of all entities in the sensor networks, and so they should be considered when defining a new type of context to be available to programmers. Contexts provide logical relations over attributes of entities at the level of the fusion data hierarchy that they are assigned.

The use of time and space operators with predicate calculus will allow ACIPL to contextually control data fusion. Contexts are the main conceptual units of the system as they are involved in controlling the state of data fusion programs as well as interpreting the attributes of data fusion. Contexts translate sensor output and serve as a language for coordinating fusion between nodes in addition to serving as a basis for associating neighboring nodes. The ACIPL system targets combining contexts and connecting programs representing different levels of the data fusion hierarchy. Because the execution of programs is controlled by contexts, programs can be considered an attribute of contexts, producing the input-output of contexts in the case of "program contexts" and producing the state of the context in general contexts. A context group forms when contexts are produced that are marked as co-fusible; the group communicates with other nodes producing the same context in an attempt to do fusion for accuracy or expanded coverage.

IV. CONTEXT APPLICATIONS

The ACIPL system explores application of context processing to distributed sensor data fusion. While previous

efforts have focused on translating sensor data into contextual representations and then providing that information to applications for aiding human-computer usability, the focus of ACIPL lies in using context to engage in data fusion. ACIPL uses context to control the system's state, to combine data across different dimensions, to interpret information, and to coordinate storage of contextually recognized objects from the fusion hierarchy.

A. *Processing*

ACIPL supports context processing through context daemons that monitor and evaluate data as contextual states.

B. *System State*

Context is explicitly used to control the system's state as variables for controlling program execution, similar to guarded commands in distributed computing. Context headers containing predicates enable execution of algorithms that in turn output attributes or objects that are contextually interpreted and shared.

C. *Addressing*

Context statements allow regions of nodes to be addressed at a time, effectively creating a context-based addressing system that allows groups of nodes to be selected, queried, and targeted.

D. *Storage*

Context-based storage and information retrieval allows similar functionality to existing systems that treat the network as a distributed database like COUGAR [4] and TinyDB [5], but with the extended power of querying across contextual relations and fusion hierarchy entities. When a sensor reads data, that data has an implicit context based on the environment and the state of the system including calibration. Understanding that context is important for post-processing of the data or transforming the data to be independent of targeted contexts. While transforming and storing data to be independent of platform characteristics is fundamental to processing sensor readings, when we arrive at systems that have fusion algorithms for high-level inference enabling self-calibration through context comparison, adding parallel hypothetical worlds processing will cause a need for explicitly storing the context that the data uses to enable further use of the data.

E. *Inference*

As outlined by the context's scope, context may be used to direct inference of sensor data for expanded range, enhanced resolution, and intelligent estimation.

V. CONTEXT TYPES

A. *Hardware Capability*

ACIPL will use Sensor Model Language (SensorML) [6] for data encapsulation and for sending hardware requirements as part of the context header. When a node receives a context header, it may check the intended platform by either ID or SensorML description of sensing types needed. By using SensorML descriptions, ACIPL integrates hardware specification into the context of the state of the network.

B. *Temporal Context*

The context of ordering in time is applicable to all levels of the fusion hierarchy due to modeling of the real world with data fusion. Time operators are used to allow the programmer to create meaningful expressions of time relations among data fusion hierarchy entities and contexts. Reference [7] focuses on the flow of time by using Propositional Temporal Logic, although other options exist like Allen's Temporal Interval Logic for study as a basis for the temporal context operators in ACIPL.

C. *Spatial Context*

Spatial context is another nearly ubiquitous property of sensor data fusion. Spatial context includes both position and orientation. While spatial position may be determined through sensors like GPS, spatial orientation may be determined through image registration techniques and relational spatial comparisons like Region Connection Calculus-8, a jointly exhaustive, pair-wise disjoint [8] set of spatial relations. By comparing context states of relational connections, the node is able to determine the orientation of its sensors relative to other nodes in order to fuse data efficiently.

D. *Data Capture Context*

Information pertaining to captured sensor data and entities in the fusion hierarchy may depend on the context in which they were created if the system has not decided between two states of a context or is operating in parallel contexts. Sensor data has the implicit context of the calibration of the system; fused entities depend on the contexts they are using for input. While the sub-contexts that serve as input to a fused entity are context properties, if branching points into parallel contexts are defined then specifying which parallel context interpretation the entity belongs to, or what contexts are used for interpreting the sensor's calibration, is used as a context type.

E. *Data Fusion Context*

The data fusion context deals with possible interpretations of the world. When a disjunction is used in a context header statement, multiple contexts may exist. In a traditional program the result of either disjunction being true is the execution of the same calculation, perhaps with side effects if the block of code depends on the conditional statement. With ACIPL, the execution of the program inherits contextual properties from the context statement; to preserve that information a disjunction in the context statement creates two separate instances based on each context.

F. *Entity Presence*

Data Fusion Contexts output data fusion objects that are passed between different levels of the hierarchies. Their execution may be conditional upon the states of multiple contexts. Their code block coordinates the output of that level of the hierarchy and is used to provide member properties of the output of the fusion object. The fusion hierarchy dictates that the attributes of a fusion level's output are the inputs to

that fusion level. For instance, a context that produces objects as output will be receiving the features of that object as input. The presence of an entity is a fundamental property that can be used in enabling other system contexts.

G. *Belief Context*

The belief context allows confidence values to be assigned to algorithm output and propagate up the hierarchy of fused entities, allowing the system to model belief networks.

H. *Custom Context*

One of the goals of ACIPL is to allow the user to create custom context types that the system can then monitor and apply to programs. Programs will contain a list of contexts and variables that are available for publishing to other programs. Custom context types provide a way to reduce network traffic by only sending the state changes that are necessary.

VI. CONTEXT PROPERTIES

When working with context in this processing model, there are a number of properties that one must take into account, including scope, level in the fusion hierarchy, states, reference frame, and number of operators.

A. *Context State*

The state of a context depends on the availability of that measurement, the default values of unrestrained contexts, and other properties. The contextual state of sensor readings depends on the surrounding contexts; the state may be augmented by constraint to other contexts' states. The state of a context may be unknown if the sensor, or context it depends on is not available. For example, contexts whose states depend of relations between entities in the data fusion hierarchy may not be available if one of the entities disappears.

B. *Data Fusion Operators*

By focusing on data fusion, contexts are separated by the type of fusion object that they produce attributes and relations over. This is to prevent mixing of contexts corresponding to different levels of the hierarchy due to changes in contexts propagating up and down the entities in the hierarchy.

C. *Context Scope and Perspective*

The scope of a context refers to whether it references the local perception of events, the local perception of events augmented with contextual information from other nodes, or group consensus. A context operating under the local perception of events only uses sensor information gathered at its own node; this represents local fusion and autonomous data gathering using local scope and local perspective. Fused data also can represent the combination of data from physically separate sensor nodes for expanded coverage or enhanced resolution; the perspective is local but the scope is now global. If group data is used but conflicts are resolved with the local data, a global scope exists with local perspective. If the contexts interpreted from the data represent the consensus formed by the group managers, then a scope is global and the perspective is global. Differentiating between scope and perspective is important when reasoning about systems with mixed local and global fusion. Although it may seem that a global context is always preferred to reference absolute position and time, local context statements may be desired in situations where nodes are severed from the network, or inside algorithms that need to guard computations based on local perspective.

D. *Context Parallelism*

Context Parallelism [9] represents the ability to operate in different possible representations of the world at the same time. The mLucid programming language [10] defines three granularities of context parallelism: fine, regular, and coarse. Fine context parallelism between program variables implies that the variables may be evaluated in parallel. ACIPL exhibits fine context parallelism between contexts at the same level of the data fusion hierarchy that do not depend on each other. Regular context parallelism implies that no contexts share global variables. ACIPL does not have fine context parallelism because input of the upper fusion levels may be dependent on lower levels of the fusion hierarchy or even at the same level. In ACIPL, the same program executing at different nodes may share input between the nodes; output for the program's group manager could be considered dependent on multiple contexts, but the group manager's output freely associates and disassociates with other nodes based on whether they are running the program, so it is not truly dependent. Coarse context parallelism is defined between independent classes of contexts that may execute in parallel; the equivalent construct in ACIPL is the context statement that combines logic from multiple types of context. Nested context statements have an order of evaluation; conjunctions of context statements may or may not exhibit coarse context parallelism.

Context parallelism in ACIPL is directly created through attaching multiple context headers to a program block so that the program will be executed in each context; after implementation we will further investigate an operator that automates context parallelism by creating a program instance for a subset of the context's states.

E. *Context Feedback*

Feedback takes the form of new contexts being introduced into the network and parameters of existing contexts being modified by contexts at a higher level. Context groups accept feedback by changing their aggregation functions' parameters, whereas individual contexts with a local perception of events may change their algorithm's parameters.

F. *State Transitions*

Contexts directly based on sensor values employ state transitions matching the natural transfer functions of the sensor readings. For contexts that are based on discrete state transitions, soft transitions could be defined with values from objects referred to in the context statement that correspond to context types in the context statement. For example, the absolute position of an object relative to another in a spatial context may be used as a soft transfer function because of the continuity of position in the real world.

VII. ARCHITECTURE

ACIPL is meant to be a middleware solution to coordinated sensing. The implementation of ACIPL is to include database connectivity for storing sensor readings and maintaining beliefs about the fused data and contexts. A context manager processes context headers, imports rules for user-defined contexts, and monitors contexts to start execution of algorithms. The context interpreter stores the output of algorithms in a context database. A group manager communicates across the network with other nodes running the same contexts and algorithms to provide the global perspective.

A. User Program Components

Each ACIPL program consists of a context header and a code block. The context header is meant to be a smaller code segment that is flooded across the network. When each node receives a context header, it checks to see if it matches the

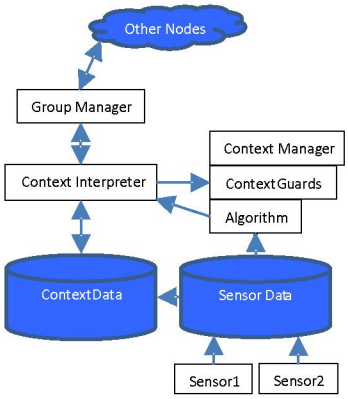

Figure 1. Organization of ACIPL components.

hardware and contextual requirements for the program before adding the context header to the list of contexts in the context manager. The context header contains the context guard statements for the program. If the sensing node does not have instructions for creating a type of context used in the context header, it requests the context executable from another node already monitoring the context. Each node monitors its context statements until conditions are met and then requests the program code block and executes the algorithm intended to be run in that context, while continuing to monitor contexts for other programs. The code block representing the algorithm is also transmitted separately from the context header to save network bandwidth for nodes where the context header does not apply or ever become true. In some instances, separate code to interpret program output as context without an intended target is included to provide history of the context over the algorithm's output. The execution of a program enables the context of the algorithm's output, which may trigger more algorithms in the network to be activated.

B. ACIPL Context Manager

The context manager interprets context headers and decides if they are valid, waiting to initiate programs in the correct context. The context manager will monitor contexts that are not currently valid but match the hardware requirements of the system and meet monitoring requirements of either time or presence of specified contexts. As shown in Fig. 1, the context manager will monitor both the algorithms and context guards that enable the algorithms.

C. ACIPL Context Interpreter

The context interpreter receives instructions from the program code block on interpreting the program's output as a context. The context interpreter then updates the context database based on the algorithm's output with the option of saving the algorithm's output with the context.

D. ACIPL Group Manager

The group manager coordinates the aggregation and consensus of sensor readings for each program. Output of the algorithm is coordinated across the network by the group manager. The group manager links to other nodes that are actively executing matching programs and synchronizes the aggregated output of each program, allowing different scopes and perspectives of the context. Contexts not running on the local node may be monitored through the group manager as well.

VIII. SUMMARY

In the near future we hope to have running an initial implementation of ACIPL using the Java programming language to enable portability, multi-tasking, and object-oriented modeling of context. When creating a parser for context statements, SensorML will be included to address sharing data and hardware capability. The initial example of ACIPL will focus on contextual data fusion of output from a blob recognition algorithm from two perspectives. We hope to enhance ACIPL with wavelet-based compression. With the initial system in place ACIPL will allow additional development of contexts to be used with data fusion.

It is important to treat context separately as an explicit object because context is implicit in many systems in assumptions that algorithm designers make about the calibration of distributed sensing elements. As systems grow, these assumptions may be transient and require different treatment based on complex system conditions. By combining these components, ACIPL provides a unique capability of programming sensor networks to assemble complex representations of the scene hierarchically.

Contextual relations between sensor data allow algorithms to link together and the network to infer relations based on templates the user writes by intersecting context operations. With specifications in the context headers, nodes are able to adapt what programs they run based on high-level statements about the scene. Shared contexts allow nodes to infer and fuse data. Separating entities according to a fusion hierarchy allow contextual detection programs to be generalized to the type of entity they operate on.

In summary, the Adaptive Context Information Processing Language provides a way of contextually controlling information fusion in sensor networks. The goal is to facilitate

sensor information fusion at different levels of hierarchy ranging from raw data up to features, objects and events.

ACKNOWLEDGMENT

We gratefully acknowledge Dr. R.L. Ewing of the Air Force Research Laboratory and Dr. Elizabeth Downie of the Dayton Area Graduate Studies Institute for their support through the AFRL/DAGSI Ohio Student-Faculty Research Fellowship Program grant No. IF07-1.

REFERENCES

[1] D. Hall and J. Llinas (editors), *Handbook of Multisensor Data Fusion*, Boca Raton: CRC Press, 2001.

[2] R. Newton, G. Morrisett, and M. Welsh, "The regiment macroprogramming system," in Proc. IEEE IPSN 2007, pp. 489-498, 2007.

[3] J. Llinas, C. Bowman, G. Rogova, and A. Steinberg, "Revisiting the JDL Data Fusion Model II," in Proc. FUSION 2004, pp. 1218-1230, 2004.

[4] W. F. Fung, D. Sun, and J. Gehrke, "COUGAR: The network is the database," in Proc. ACM SIGMOD 2002, pp. 621-621, 2002.

[5] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: an acquisitional query processing system for sensor networks," ACM Trans. Database Sys., vol. 30, pp. 122-173, March 2005.

[6] M. Botts (ed.), "OpenGIS Sensor Model Language (SensorML) Implementation Specification, Version 1.0.0, OGC07-000," OpenGIS Consortium (OGC), University of Alabama in Huntsville, 2007.

[7] F. Wolter and M. Zakharyaschev, "Spatio-temporal representation and reasoning based on RCC-8," in Proc. of Principles of Knowledge Representation and Reasoning, KR2000, pp. 3-14, 2000.

[8] A. G. Cohn, B. Bennett, J. Gooday, and N. M. Gotts, "Representing and reasoning with qualitative spatial relations about regions," in *Spatial and Temporal Reasoning*, O. Stock, Ed. Dordrecht: Kluwer Academic Publishers, 1997, pp. 97-134.

[9] M. A. Orgun and W. Ma, "An overview of temporal and modal logic programming," in Proc. ICTL '94, pp. 445-479, 1994.

[10] W. Du, "Context parallelism in an indexical programming language," in Proc. IEEE Int. Conf. on Computing and Information, pp. 235-239, 1993.