ROSE 2008 - IEEE International Workshop
on Robotic and Sensors Environments
Ottawa - Canada, 17-18 October 2008

TOWARDS A MODEL AND SPECIFICATION FOR VISUAL PROGRAMMING OF MASSIVELY DISTRIBUTED EMBEDDED SYSTEMS

Meng Wang, Varun Subramanian, Alex Doboli
Department of Electrical and Computer Engineering
State University of New York at Stony Brook, Stony Brook, NY 11794-2350

Daniel Curiac and Dan Pescaru
Faculty of Automatics and Computers
"Politehnica" University Timisoara, Romania

ABSTRACT
Massively distributed embedded systems are rapidly emerging as a key concept for many modern applications. However, providing efficient and scalable decision making capabilities to such systems is currently a significant challenge. This paper proposes a model and a specification language to allow automated synthesis of distributed controllers, which implement and interact through formalisms of different semantics. The paper refers to a case study to illustrate the main capabilities of the proposed concept.

Keywords
Specification languages, massively distributes systems, goal-oriented, scalability, predictability.

1. INTRODUCTION
Massively distributed embedded systems are rapidly emerging as a key enabling concept for many modern applications in environmental monitoring, energy conservation, healthcare, infrastructure management, homeland security, manufacturing, and many other [1, 2, 3, 7, 9, 14]. This is due not only to sensing and electronic devices becoming extremely cheap and small in size (thus, deployable in large numbers) but also to the potential of having significantly superior decision making quality if related problems are tackled together instead of separately. Moreover, this helps improving the robustness of the systems as experience has shown that many disasters occur due to unaccounted correlations between the sub-systems. Providing reliable and efficient decision making capabilities to massively distributed embedded systems currently represents a main challenge [16].

The envisioned decision making paradigm is different from existing approaches, which either focus on centralized control or on local control [16], many times using ideas inspired from social or biological systems. Centralized control is well understood and reliable but does not scale well for large systems. In contrast, local control works well for large systems but with the exception of simple situations, its overall performance is hard to be captured. We argue for a distributed decision making strategy in which parameterized control procedures implementing different strategies are dynamically introduced or removed depending on the specific optimization goals and operation conditions of the application. The controllers operate according to different models depending on the nature of their decision making. For example, reactive controllers use physical inputs for local control but operate under the constraints set by more "abstract" decision making procedures, which analyze broader situations based on aggregated (abstracted) data and procedures. The proposed decision making mechanism is flexible, scalable, and predictable. A key component of the approach is a suitable model and specification notation for massively parallel applications.

This paper presents a novel control model and the related specification for developing massively distributed embedded applications. We argue that a main challenge in providing scalable descriptions for such applications is due to the wide variety of interactions that emerge among the composing sub-systems, some of which are hard to anticipate a-priori, or might change their importance dynamically during execution. The proposed solution is to describe the nature of interactions that might occur among modules while leaving to task of optimally implementing these interactions to the compiler and execution environment.

More specifically, the proposed model defines the operation goals of each sub-system (e.g., the criteria to be maximized or minimized during execution) and the physical capabilities of a module to achieve a certain goal (such as its maximum processing speed, highest bandwidth, etc). Different interaction types are introduced depending on the way the sub-systems influence each other their goals and capabilities. The specification is compiled into a network of Decision Modules (DMs), which use reactive models to control decisions at the physical level, and more abstract formalisms, such as Task Graphs and Markov Decision Processes, to perform broader, more strategic decisions. The multi-semantic DMs interact with each other through *constraints* by which the strategic modules restrict the reactive DMs to guarantee satisfaction of the global goals, and *feedback* offered by the reactive DMs about the feasibility of the constraints.

Section 2 summarizes the related work. Section 3 defines the distributed control model and Section 4 introduces the main language constructs. Section 5 illustrates the design of an application. Finally, conclusions are offered.

2. RELATED WORK

The concept of Visual Programming (VP) was arguably proposed in the 80s [5], however, it is only recently that its advantages for embedded applications became apparent. VP languages have been proposed for applications like managing smart oilfields [7, 12], vehicle tracking [4, 6], contour finding [6], environmental monitoring [2], etc. Some of the related work is presented next.

Region Streams [6] is a functional macroprogramming language for sensor networks. Specification is based on successive filtering and functional processing of data pools. The data model is based on continuous data streams sampled from the environment and groups of nodes defined by their specific interests in space and over time. Language constructs enable aggregation of the data streams from a region and application of a function to the streams in a region. Abstract Task Graphs [2] is also a functional specification in which tasks sample from and place data into pools. There is no other interaction type between tasks. Channels are filters for associating only specific data from a pool to a task. Tasks are executed periodically or when input data is available.

Semantic Streams [9] implements a query-based programming paradigm, which fits well applications in which sensor networks operate as large distributed databases. Queries formulated as logic programs are converted by the compiler into a service graph for the network. Data sensing is modeled as streams. Other constructs include filtering by specifying properties of the streams, defining regions and sub-regions of the physical space, and performance requirements (e.g., quality of service). Kairos [4] proposes a set of language-independent extensions for describing global behavior of sensor networks controlled centrally. The extensions assume shared memory to allow any node to iterate through its neighbors and address arbitrary nodes.

Similar to [6, 9], the data model of the proposed specification language is based on data pools and continuous data streams to the modules. However, it differs in that it focuses on optimal decision making in massively distributed environment, and not on algorithmic descriptions. Therefore, we argue that the language is orthogonal to the existing notations as it concentrates on the interactions between groups of nodes, or nodes and environment, and less on the behavior of the individual nodes.

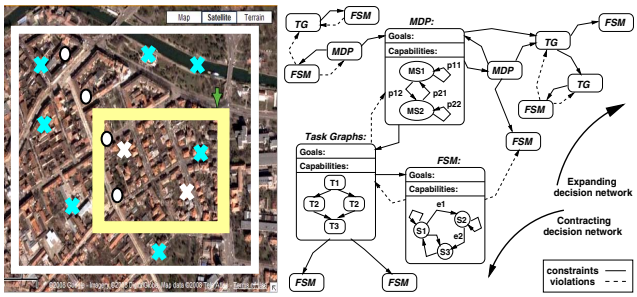

Figure 1: Specification and distributed control concept

3. GOAL-ORIENTED MODEL

Figure 1 illustrates the main characteristics of the proposed distributed control concept (right figure) and the related specification language (left figure). The wide geographical areas from which data is sampled define pools of heterogeneous data (e.g., temperature, humidity, number of vehicles moving, etc.). The association of data acquisition to the physical area is defined graphically, as shown in the left figure by the larger and smaller rectangles. Moreover, for each defined area, the user specifies the *goals* that ought to be achieved by the distributed application, such as maximizing the traffic flow through the area, or keeping the pollution level below a predefined limit.

The execution platform is a massively distributed network of embedded controllers. Each controller node comprises hardware for sensing and actuation as well as processing, storage, and communication. In addition, hardware is reconfigurable, so that the available pool of resources at a node can be configured to meet different performance trade-offs, like variable processing speed, energy consumption, resolution, storage space, and so on. Hence, the model assumes that the functionality (algorithm) of each node is well defined and fixed but the node's performance is parametric and dynamically changing at execution.

The following three aspects are the core of the proposed model:

- *Separation of algorithmic aspects from goals*. Algorithms describe the ontology of the application, and, over time, remain the same for all nodes in the network. In contrast, goals define optimization criteria, performance, safety, etc., depend on the specific execution platform and conditions, and change dynamically in time. While building algorithms is arguably more efficiently done by humans, finding the parameters for optimal execution is cumbersome but can be automated.
- *Description of interdependent, heterogeneous sub-systems.* Global goals in large applications are likely to transcend different sub-system types. Moreover, the significance of the various related components can change over time, which can invalidate any static interaction scheme between sub-systems.
- *Avoiding explicit descriptions of synchronizations and data transfers.* Explicit specification of inter-node communication reduces scalability and reusability. It is hard and unreliable to attempt capturing all possible interactions between the components of massively large scale applications. Instead, the design environment ought to identify the best interaction schemes between components, so that the overall goals as well as the goals of the modules are met.

The proposed *goal-oriented* model comprises of separate decision modules (DMs) that operate to optimize a well-defined set of goals while the overall goals of the application are also being optimized. Each module executes a set of parameterized behaviors (algorithms) for which the parameters are automatically computed based on the information provided through the goal-oriented descriptions. DMs interact with each other only in small numbers but can perform global decisions by using data aggregated from large regions and by strategic decision methods (e.g., based on Markov Decision Processes). Figure 1 (right) illustrates a network, which comprises different decision making models (Finite State Machines, Markov Decision Processes, etc.).

Similar to other specification languages for sensor networks, the proposed data model is based on *data pools* associated to regions and groups. Modules sample inputs from and generate outputs to a data pool for region. Regions represent continuous collections of tokens, such as a geographical area. Groups are discrete collections of tokens. Regions and groups can be associated to a specific physical area of the environment, or can be described by their defining properties (see Figure 1).

Efficient and robust decision making must be guaranteed for hard-to-predict conditions. The overall behavior should be capable of

autonomously meeting performance requirements, e.g., real-time constraints, bandwidth limitations, energy constraints, speed requirements, precision, etc. The underlying decision making model (DMM) is based on a *multi-semantic decision making networks* that represents the overall application performance at different levels of abstraction and using different evaluation formalisms. The low levels employ reactive models, e.g., finite state machines, which are capable of tackling unexpected situations. The upper levels use less flexible models but with more predictable performance, such as data flow graphs and task graphs. This way, the semantic hierarchy offers a smooth transition from the fully reactive behavior at the embedded node level and the deterministic operation at the application level. The number of abstraction levels and the decision making models for each level depend on the application.

The interaction between the different semantic models is achieved through *top-down* and *bottom-up constraint transformation* along the entire multi-semantic network: the top-down mechanism constrains the lower decision making levels by bounds introduced for their goals. As long as the low level operation stays within the bound it can be guaranteed that the overall performance is satisfied. The bottom-up mechanism signals when constraints (imposed by the upper levels) are unsatisfiable for the lower levels. Figure 1 (right) shows a semantic hierarchy with three models, the bottom layers represent reactive behavior, and the top layers offer a performance predictive description of the system as Task Graphs (TGs) with data dependencies and Markov Decision Processes (MDPs). The scheduling results of TGs are used to compute timing constraints for the reactive behavior of the bottom level DMs. As long as their reactive operation stays within these constraints, the overall timing requirement can be guaranteed. Bottom-up constraints express the amount of performance "violation" occurred at the physical level, and ought to be considered when recomputing the decision at the upper levels. The propagation of top-down and bottom-up constraints is performed continuously at runtime.

The networked decision making in Figure 1 (right) operates as follows. The reactive DMs employ input sampling and event driven decision making mechanisms, such as Finite State Machines (FSMs). The reactive behavior switches from one task to another depending on the occurrence of events. For example, the system switches from $\mathbf{S_1}$ to $\mathbf{S_2}$, if event $\mathbf{e_1}$ occurs. Each embedded unit executes its own controller. Hardware reconfiguration offers the capability of selecting online task implementations with different parameters, like speed, energy, memory, communication bandwidth, etc. Several individual controllers might decide to collaborate by building collectively a shared description that defines how the individual controllers use jointly any shared resources to achieve common objectives, such as the access over time (schedule) of vehicles accessing intersections. For example, the shared description can be a Conditional Task Graph (CTG) [17], which includes decisions specific to different operation conditions, such as various traffic loads. Please note that CTG decision making is at the level of small local areas rather than individual controllers. Next, the CTG descriptions of the correlated areas are used to build description covering broader areas, such as those at the *Markov Decision Process* level. This step distributes the overall goals into goals for the individual sub-systems using information from the Markov Decision Process level.

The execution environment determines the structure of the distributed multi-semantic decision network as well as the node's performance parameters. Also, the network expands and shrinks depending on the nature of the application. A case study for different decision making procedures is presented in Section 5.

4. SPECIFICATION CONSTRUCTS

The main constructs of the proposed goal-oriented specification notation are illustrated in Figure 2. The basic specification entity is a Decision Module (DM), which can correspond to local points, physical areas, zones, or larger regions. As explained in Section 3, each DM executes specific application-specific algorithms related to sensing, processing, and actuation. This part is internal to each DM and is specified using an existing programming language, like C++ or Java. The focus of the proposed specification notation is on describing the local and global goals as well as the nature of interactions between modules. The actual interaction mechanism, such communication mechanism and parameters, is produced automatically by the compiler. For simplicity, the section offers a descriptive presentation instead a formal one.

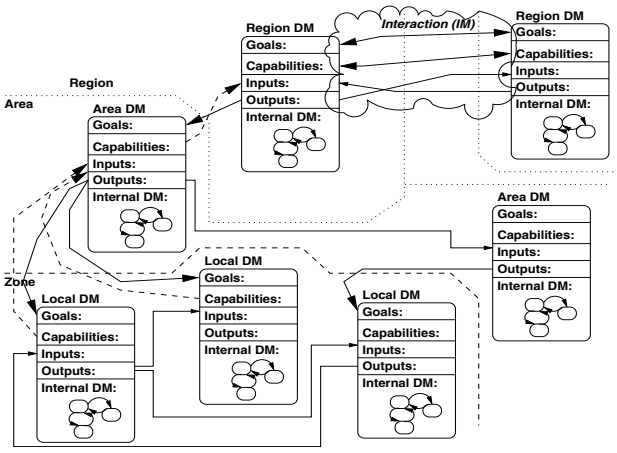

Figure 2: Specification for scalable decision making

Figure 2 illustrates that a DM has four main parts: inputs, outputs, capabilities, and goals. The four parts are detailed next.

Inputs. DM inputs either sample the pool of data associated to a physical region or are interaction data coming from other DMs. The acquisition semantics of an input can be continuous time, discrete time, or event driven. Moreover, inputs can refer only to certain facets of physical signals, e.g., voltage, current, phase, and frequency of a signal. The user can also specify aggregation of signals over time and space (integrals for continuous inputs and sums for discrete signals) and rate of change (sensitivities) with respect to time, space, or other signals (derivates for continuous inputs and differences for discrete inputs).

Outputs. DM outputs relate to the outputs produced by the algorithm of a module. Outputs include physical actuation and control signals.

Capabilities. Each module has a limited set of physical capabilities, such as the highest amount of service requests it can service, local memory, processing speed, energy resources, communication distance and bandwidth, and so on. These capabilities affect a nodes ability to maximize its local goals, and also percolate in influencing the quality of the optimal value that is reached for the overall application. For example, a node's low processing speed or lack of memory might restrict the amount of

alternatives that are locally analyzed, and hence affect the quality of the optimization process conducted by the node.

The user can define attributes for capabilities, like upper bound, lower bound, and average value in time and space of a capability.

Goals: The part indicates the goals to be optimized by each DM. Goals can be expressed either as maximizing or minimizing a cost function, or as a constraint satisfaction requirement, in which the goal expression must either exceed or fall below a threshold value. The cost function is defined over outputs and capabilities.

Components interact through interactions. The following kinds of interactions can be distinguished in the proposed notation:

A. Collaborative interactions: Collaborative interactions are set up between DMs, which have non-conflicting (non-competing) goals. Modules interact with each other through inputs and outputs, e.g., one module produces outputs to the pools that serve as inputs to another module. The goals and capabilities of the modules are not affected by this kind of interactions.

B. Competing interactions: Interactions are set up between DMs with competing goals, like optimizing one goal affects adversely the optimization of the other. Similarly to collaborative interactions, modules interact through their inputs and outputs but cannot change their goals or capabilities.

C. Guiding interactions: Guided interactions are between DMs at consecutively higher levels in the semantic hierarchy. DMs at upper levels generate outputs that are used to set the goals of the modules at lower levels. This way the first modules are steering the goals and hence the behavior of the latter modules.

D. Enabling interactions: Enabling interactions are information transfers from lower (reactive) semantic levels to the upper levels. A lower DM transmits information about its capabilities to an upper DM, so that the second can use this knowledge during a decision making that might affect the goals set for the lower DM through guided interactions. Enabling interactions are the information links through which upper DMs acquire knowledge about the actions that are ongoing in the real world.

In the proposed specification notation, the nature of interactions is dynamic and does not explicitly state the DMs participating to it. Instead, the user describes the conditions under which DMs interact with each other, like the change of an input, exceeding a threshold value, exceeding certain capabilities, etc. Interactions are described using Interaction Modules (IMs). IMs define any data transformation (such as data aggregation, filtering, etc.), which is required if DMs of different formalisms are interacting. Similar to DMs, IMs have goals and capabilities. Moreover, the user must specify for each of the four interaction types, the formal mechanism (TGs, MDPs, etc.) used to resolve the interaction.

5. CASE STUDY AND INSIGHT INTO SPECIFICATION COMPILING

The case study refers to coordinated environmental monitoring (air quality and temperature) and traffic management in urban areas. Please note that the two sub-systems can operate either stand alone or in relationship.

A.*Monitoring*. The area monitored for pollution is defined statically by indicating the physical region (e.g., the large box in Figure 1). The position of three air quality sampling nodes is shown in the figure as darker crosses, but a much larger number of sampling nodes is used to uniformly cover the entire area. The second area is the small rectangle, which indicates the area monitored for temperature. Two of the related temperature-sensing nodes are shown as white crosses in the figure. Figure 3 shows the DMs for the graphical description in Figure 1.

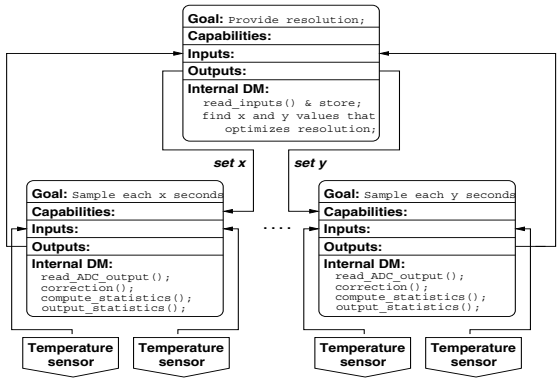

Figure 3: Specification for environmental monitoring

B.*Traffic management*. Traffic management, the second sub-problem, requires that the total number of vehicles passing though the entire area (of the figure) is minimized while the delay of any car is kept under a predefined limit.

Figure 4 shows the simple traffic management situation. Figure 4(top) presents a street network, in which arcs indicate the flow of traffic. The goal of decision making is to maximize the traffic flow that passes through the area in a given time **T**.

Traffic lights (shown as black bubbles) are positioned at each intersection, and each is controlled by its local DM called ITU (Intersection Traffic Unit). Neighboring ITUs interact to form ZTCU (Zone Traffic Coordination Unit) that coordinates their operation. IMs are defined such that ITUs interact as long as there are cars passing through an intersection but their total number is below of limit indicating traffic congestion. Moreover, ZTCUs of an area interact and are coordinated by a higher level DM called ATCU (Area Traffic Control Unit). Finally, ATCUs are supervised by even more abstract DMs called TCU (Traffic Coordination Unit). According to the model in Section 3, ITUs, ZTCUs, and ATCUs can be statically defined, or can emerge dynamically during operation based on the actual traffic patterns and the best way of coordinating the traffic lights.

Figure 4(top) illustrates the two lower levels of the multi-semantic network: the FSM level and the Conditional Task Graph (CTG) level. The figure also shows the interaction between the two levels. Each of the ITUs in the figure implements a FSM with three states (red, yellow, and green). These DMs are run by each embedded node. Assuming cyclic traffic signals, arcs indicate state transitions which are taken after the FSM spends $\Delta\mathbf{T}_{i,j}$ time units in the current state (the time is called *split* time). The time intervals are specific to each ITU and each transition, and are fixed by the ZTCU for optimizing the traffic flow. The schedule of the traffic signal repeats over a period $\mathbf{T}$ (called *cycle time*). Neighboring ITUs interact with each other to decide the delay between the monitored intersections (the delay is called *offset*).

The ZTCU behavior is expressed as a Conditional Task Graph (CTG) [17], which indicates the activities performed by two cars arriving at ITU_1 and ITU_2 in the figure. Activities include passing through the intersection, moving to the next intersection, and so on. Each of the activities is characterized by a typical execution time $\mathbf{T}^{ex}_{k}$, which is estimated based on the data collected locally

from the sensors at the traffic lights. ZTCUs set the cycle and split times of each intersection, and also find the initial delay values. Execution times differ for different traffic scenarios, e.g., light traffic (indicated as branch **L**) and heavy traffic (shown as branch **T**). The semantics of the application states that one and only one of the branches $\mathbf{L_i}$ and $\mathbf{H_i}$ is executed for a traversal of the graph. This semantics defines a conditional (multimode) behavior.

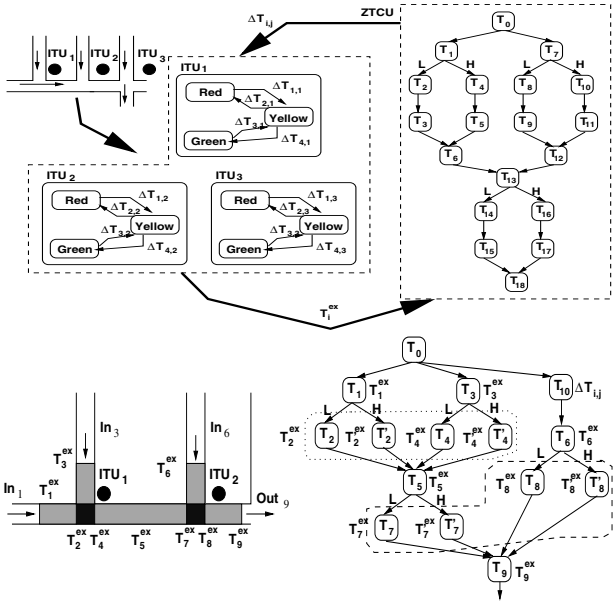

Figure 4: Decision making using semantic hierarchy

Figure 4(bottom) presents the concrete CTG for a zone with two consecutive intersections controlled by the traffic signals ITU_1 and ITU_2. The graph presents the activities occurring during one cycle of the entire area, even though the cycle might include several cycles of the individual traffic signals. The process is repetitive after each area cycle. In this example, we assumed single-lane roads, which means that two cars coming through the intersections ITU_1 or ITU_2 cannot move simultaneously. Hence, certain road sections (shown as dark areas in the figure) act as shared resources for two or more cars, while other roads are dedicated resources.

The model in Figure 4(bottom) defines tasks $\mathbf{T_1}$ and $\mathbf{T_2}$ as being the travel of a vehicle entering the area from the left, tasks $\mathbf{T_3}$ and $\mathbf{T_4}$ as representing the moving of a vehicle coming from the top-left road, tasks $\mathbf{T_6}$ and $\mathbf{T_7}$ represent a car moving from top right, and so on. The resulting CTG is illustrated in the figure. Each task has two attributes: $\mathbf{N_i}$ is the number of vehicles that traverses the section during one cycle, and $\mathbf{T_i^{ex}}$ is the time required for the vehicles to pass through the section. Obviously, the two attributes are related to each other. In the graph, the tasks sharing the same resources (the road sections are depicted in dark) are marked with dashed and dotted lines, respectively. In addition, the dummy task $\mathbf{\Delta T_{I2}}$ represents the offset time between the two traffic signals.

The ZTCU for the area computes online the optimal scheduling of the tasks $\mathbf{T_i}$ while tackling requirements such as, (i) maximize the total number of vehicles passing though the zone, (ii) minimize the time taken to cars to move through the zone, (iii) minimize the fuel consumption in a certain area, (iv) minimize the amount of polluting gases that are generated, and so on. To avoid continuous recomputing of schedules and to improve the flexibility of the approach, the ZTCU scheduler calculates traffic signal schedules for different traffic conditions, which determine the "behavior" (and thus the mathematical models) for the task attributes, such as parameters $\mathbf{N_i}$ and $\mathbf{T_i^{ex}}$.

Please note that each computed schedule corresponds to traffic conditions described by conditional behavior and the qualitative parameters $\mathbf{L_i}$ and $\mathbf{H_i}$ in Figure 4. For example, if the traffic load is low on the segment **1** then the branch labeled **L** is taken after the task $\mathbf{T_1}$ is completed, hence task $\mathbf{T_2}$ is performed. If the traffic load is high then the branch labeled **H** is selected, and the task $\mathbf{T_2'}$ is pursued. Tasks $\mathbf{T_2}$ and $\mathbf{T_2'}$ can never be performed simultaneously as they correspond to mutually excluding conditions (e.g., the traffic intensity cannot be simultaneously low and high). This generates 16 different situations for the case study in Figure 4(b), such as schedule $\mathbf{S_{(L,L,L)}}$ with the total time $\mathbf{T^{tot}_{(L,L,L)}}$ for the situation in which the traffic density is light through all three intersections. The actual conditions, which are labeled as **L** and **H**, can be different for different traffic lights, and can involve more than two situations. The concrete task scheduling for CTGs can be computed with algorithms similar to the one in [17].

Figure 5(top) illustrates the interaction between the local ITUs (FSMs) at each traffic signal and the CTG scheduling at the ZTCU. The scheduling table computed online by the ZTCU is shown in the figure. The table includes the scheduling solutions for different scenarios that might arise in traffic. The table presents the schedules for two situations: one in which traffic is light along all directions, and the second in which the traffic along direction two is more intense, while the others remain low. The traffic schedules are computed so that the total number of cars passing through the zone is maximized while the timing along different directions is kept within fixed deadlines (e.g., to avoid large delays along certain directions due to enabling heavy traffic along other directions). The boxes in gray indicate some of the idle times along a direction.

Please note that the actual semantics of the conditions $\mathbf{L_i}$ and $\mathbf{H_i}$ (e.g., what traffic conditions actually define them) are refined during execution depending on the variation of the parameters $\mathbf{N_i}$ and $\mathbf{T_i^{ex}}$. More conditions allow more precise tuning of the schedules, however, the computational complexity of calculating the optimized schedules is also higher [17].

The information in the CTG scheduling table is mapped down onto the FSMs of each individual traffic signal. For example, for Scenario **1** (in which traffic is low along all directions), the time required for performing task $\mathbf{T_3}$ plus the following idle time generate the timing constraint $\mathbf{\Delta T_x}$, which states that after that amount of time the FSM must be in state **Green**, thus allowing traffic to progress along direction two. Similarly, for Scenario **2**, the time for task $\mathbf{T_1}$ plus the following idle time define the constraint $\mathbf{\Delta T_y}$, which states that the traffic signal ought to be in state **Red** after $\mathbf{\Delta T_y}$, which allows traffic to progress along direction one. As long as each traffic signal meets its local timing constraints ($\mathbf{\Delta T_x}$ and $\mathbf{\Delta T_y}$), the behavior at the ZTCU level still meets the traffic schedules specified in the ZTCU table. Please note that the local FSMs can switch more often: if no vehicle is traveling in one direction and a single car is approaching from another direction, but for "pools" of vehicles the signaling must follow the timing constraints of the ZTCU table.

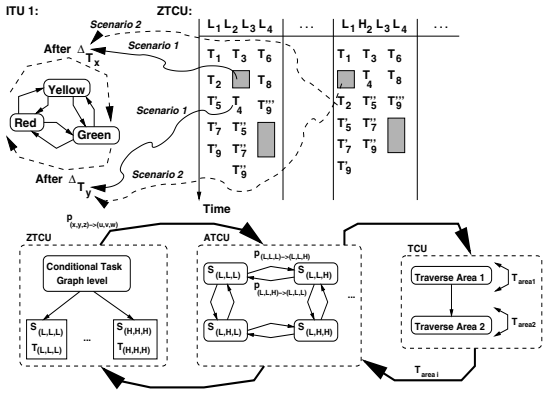

Figure 5: FSM - CTG – MDP – Functional Graph network

Figure 5bottom) presents the upper levels of the hierarchy, including the Markov Process (ATCU) and the Functional Graph (TCU) levels, and their connection to the lower Conditional Graph level (ZTCU). For an estimated number of vehicles that pass through the area, the ZTCU has the optimal activity schedules for each estimated traffic scenarios. If the timing delay $\mathbf{T}_{area\_i}$ of a ZTCU schedule exceeds the necessary timing constraint set for the area traversal, then the traffic light controllers are adjusted so that the new traffic scenarios has a schedule that meets the constraint.

6. CONCLSION

Massively distributed embedded systems are rapidly emerging as a breakthrough concept for many modern applications. However, providing efficient and scalable decision making capabilities to such systems is currently a significant challenge. The paper proposes a model and specification language to allow automated synthesis of distributed controllers, which implement and interact through models of different semantics. Scalability of descriptions is realized through defining the nature of interactions that can occur among decision modules while leaving to task of optimally implementing these interactions by the execution environment. The notation defines the operation goals of each sub-system (e.g., the criteria to be maximized or minimized during operation) and the physical capabilities of a module to achieve a certain goal. Different interaction types are introduced depending on the way sub-systems influence each other their goals and capabilities.

The paper refers to a case study to illustrate the model and the related decision making steps. Compared to similar work, the proposed model and specification notation differs in that they focus on optimal goal satisfaction in massively distributed systems and not on algorithmic descriptions. Therefore, the language is orthogonal to existing notations as it concentrates on the interactions between groups of nodes, or nodes and environment and less on the behavior of the individual nodes. This simplifies specification and helps scalable decision making.

REFERENCES

[1] Batini, C. et al. Visual languages and quality evaluation in multichannel adaptive information systems, Journal of Visual Languages & Computing, 18 (2007), 513-522.

[2] Bakshi, A, Prasanna, V. et al. The Abstract Task Graph: A Methodology for Architecture-Independent Programming of Networked Sensor Systems, Proc. EESR (2005).

[3] Camara, K., Jungert, E. "A visual query language for dynamic processes applied to a scenario driven environment", Journal Visual Languages & Computing, 18 (2007), 315-338.

[4] Gummadi, K. et al. Macro-programming Wireless Sensor Networks using Kairos, Proc. Int'l. Conference on Distributed Computing in Sensor Systems (2005).

[5] Johnston W. et al. Advances in Dataflow Programming Languages, ACM Computing Surveys, Vol. 36, No. 1, (March 2004), 1-34.

[6] Newton, R., Welsh, M. Region Streams: Functional Macroprogramming for Sensor Networks, Proc. Workshop on Data Management for Sensor Networks (2004).

[7] Soma R. et al. A Semantic Framework for Integrated Asset Management in Smart Oilfields. IEEE Symposium on Cluster Computing and the Grid (2007), 119 - 126

[8] Wache, H et al. Ontology-Based Integration of Information – a Survey of Existing Approaches, Proc. IJCAI--01 Workshop: Ontologies and Information Sharing (2001).

[9] Whitehouse, K., Zhao, F., Liu, J. Semantic Streams: a Framework for Declarative Queries and Automatic Data Interpretation, Technical Report, Microsoft Research, MSR-TR-2005-45 (2005).

[10] Yetagnon, K. et al. A Web-centric semantic mediation approach for spatial information systems, Journal of Visual Languages & Computing, Elsevier, 17 (2006), 1-24.

[11] Yu, Y. et al. On Communication Models for Algorithm Design in Networked Sensor Systems: A Case Study, Pervasive and Mobile, 1 , Issue 1 (2005), 95 – 121.

[12] Zhang, C., Bakshi, A., Prasanna, V. ModelML: a Markup Language for Automatic Model Synthesis, IEEE Conf. Information Reuse and Integration (2007), 317-322.

[13] Microsoft Visual Programming Languages (VPL), http://msdn2.microsoft.comen-us/library/

[14] Bakshi. A, Prasanna, V., Ledeczi, A. MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems, ACM SIG-PLAN Notices (2001).

[15] Lange, C., Wijns, M., Chaudron, M. Supporting task-oriented modeling using interactive UML views, Journal of Visual Languages and Computing, 18 (2007), 399-419.

[16] Passino, K. Biomimicry for Optimization, Control, and Automation, Springer (2005).

[17] Eles, P. et al. Scheduling with Buss Access Optimization for Distributed Embedded Systems, IEEE Transactions on VLSI Systems, Vol. 8, No. 5 (Oct 2000), 472-491.