Design and Verification of Cellphone-based Cyber-Physical Systems: A Position Paper

Rodion Podorozhny
Computer Science Department
Texas State University
San Marcos, Texas 78666
Email: rp31@txstate.edu

***Abstract*—This paper outlines an approach to explore methods for design and verification of cellphone-based cyber physical systems. The use of cellphones for development of such control systems has a number of benefits. Cellphones are relatively cheap, they already combine a number of sensors and communication capabilities that make them suitable candidates for relatively inexpensive embedded systems. At the same time there are certain peculiarities and restrictions in the existing operating systems and programming environments for cellphones that do not allow transferring existing methods for design and verification of control systems directly. We suggest an approach to explore such methods adapted for cellphones as embedded devices.**

I. Introduction

As cellphones computational power, sensor array and capabilities of cell networks increase there is a growing trend for using the cellphones as embedded devices for controlling technical objects. The vast majority of existing projects focus on low level stabilization control. They do not venture into application of higher level problem solving methods (real time distributed planning as in [5], statistical machine learning, goal oriented action planning). Nor do they provide generalized model driven methods for design of such systems. To the authors knowledge there is no reported experience in verification or systematic testing applied to control systems implemented on the basis of cellphones. It is very likely that in the near future there will be a growing demand for methods of design and verification of such systems. Thus this problem is important. This paper suggests an approach to explore suitability of existing architectures, algorithms, and generalized methods for design and verification of real time distributed control systems for teams of mobile vehicles (cyber-physical systems) controlled by cellphones.

II. Motivation

Imagine being able to download an application to a regular cellphone, plug it into an on-board control system of some technical device or vehicle and turn it into an easily reconfigurable inexpensive robot. At this time there is already a growing community of enthusiasts who write customized cellphone applications of this kind (e.g. cellbots.com). The AI and aerospace communities have also produced a number of inexpensive control systems for ground and aerial vehicles that are based on boards such as Arduino and Beagle (ArduPilot, Paparazzi [2]). Some of these control systems have been put to very good use on mini UAVs for the purposes of meteorological or ecological studies. There are also examples of the use of cellphones as controllers for industrial assembly lines. Definitely there is a large area of applications for inexpensive robotic systems. We propose to explore adaptation of software engineering methods for design and verification of control systems designed to run on regular cellphones with as little customization as possible. In addition, we want to augment such control systems with reasoning capabilities. Most of available control systems of this kind focus only on the low level stabilization control. A direct transfer of existing software engineering methods is complicated by the peculiarities of programming environments and operating systems for cellphones. By engineering methods we understand methods for synthesis and analysis of software. In particular, model-driven approach to development similar to the use of SIMULINK for C-based control systems, verification methods of the model, verification and testcase generation methods for the source code and executables. Without the loss of generalization we will focus on Android based cellphones as it is less hassle to install new applications on them and it is easier to modify the Android operating system itself because its source code is publicly available.

Currently we see the following difficulties with direct application of existing software engineering methods:

- The cellphone programming environments are event-based. A cellphone application must subclass classes from an existing framework and override methods invoked in response to predefined events. This puts a sort of a straight jacket on the architecture of a concurrent control system, though still, some variety is possible. Besides, modern verification tools are not suited for verification of event-based system of this kind. For instance, NASA's Java Pathfinder model checker [1] cannot be used for verification of Java code written for an Android platform to check for deadlocks, race conditions or domain specific properties
- The cellphone operating systems are not real-time. Our goal is to avoid deep modifications to the operating system (such as Robot Operating System (ROS)) and the virtual machine (VM). For instance, for Java, real time VMs do exist (JamaicaVM, SimpleRTJ). They are either very expensive or do not support the full range of libraries. We do need to use some architectural solutions to ensure the required frequency and timeliness of sensor measurements and control outputs.

COLLABORATECOM 2013, October 20-23, Austin, United States

DOI 10.4108/icst.collaboratecom.2013.254262

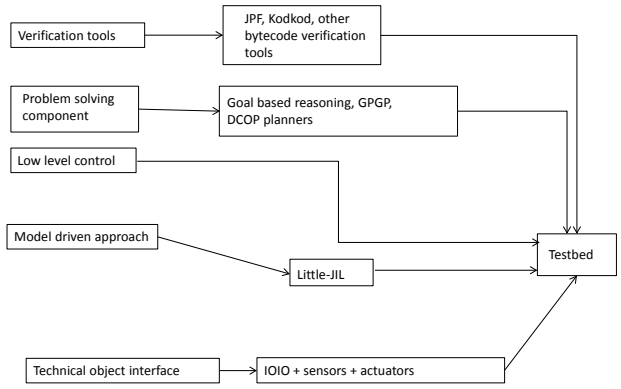

Fig. 1: Approach

- Modern model-driven environments for control systems such as SIMULINK generate C source code and are not directly applicable to cellphone applications
- Modern model-driven environments do not enable one to define reasoning software components (e.g. goal oriented behaviors, hierarchical task networks for distributed planning).

In the next section we will outline proposed solutions to these difficulties.

III. Approach

A roadmap of proposed solutions is depicted in Figure 1.

A. Low-level control

So that to add to the body of knowledge about applicable architectures for cellphone-based control systems we would like to do comparative analysis of the following architectures based on performance, ability to satisfy real time constraints, reliability, ease of verification :

- completely reactive, the control code will be invoked in response to the change of readings of sensors that are either embedded in the cellphone or attached to it
- control system without internal concurrency driven by a single active control loop
- control system with internal concurrency, organized to match the structure of the technical vehicle (i.e. a thread per a group of sensors and actuators responsible for control along a certain axis or grouped according to some other logical criterion). Thus it will be a set of concurrent communicating control loops.

B. Problem solving component

At first we would like to evaluate the Generalized Partial Global Planning [5] applied to definition of coordination among the technical objects or vehicles. Individually, vehicles will use goal oriented action planning whose action choices can be overridden as a result of negotiation between the vehicle control systems according to the GPGP protocols. In future we would like to explore algorithms for solution of Distributed Constraint Optimization Problem (e.g. as in [4]) instead of GPGP and evolutionary neural networks [7] to augment the goal oriented action planning.

C. Model-driven development

As a language for model-driven development we would like to adopt the Little-JIL [8]. It will be used to define the bodies of control loops. In this application each leaf step will correspond to a certain fragment of code responsible for reading a sensor, processing sensor data, calculation of actuator response, outputting response to actuator. Besides we will use this process language for definition of hierarchical task networks for the GPGP. Thus it will be possible to reconfigure both the low level stabilization and problem solving component.

D. Verification techniques

We will also be exploring various methods for verification of the resultant control system. For instance, we will explore a method for creation of a software mock-up of the control system that can be verified with Java Pathfinder. We will also extend the method for verification of multi-agent protocols [6] to verification of cooperative behaviors of cyber-physical systems, taking into account the physical states of the controlled systems they end up with after following the negotiation protocols.

IV. Architecture

This section will describe the main components of the proposed concurrent architecture. Figure 2 depicts these components and interactions between them. To highlight the possible relationships between the components we use a custom architecture description language. The meaning of the relationships is defined in Figure 2. Let us start the architecture description. The control system reacts to reception of assigned tasks. Tasks are identified by their names. A task decomposition storage contains the decompositions of tasks that a control system can perform. For instance a process language such as Little-JIL can be used to define these task decompositions.

An instance of a task planner is created in response to a task assignment. Task requests can come in before the previously assigned tasks are completed or aborted. Thus several instances of task planners can run concurrently. The decomposition of the task planning itself is depicted in Figure 3. This figure uses a simplified function decomposition process language along the lines of Little-JIL. The edges correspond to functional decomposition, the nodes to steps. Various execution constraints are defined via sequencing notation: seq (sequential), cond (conditional), and sum (no order defined, though completion of all substeps is needed). This is a subset of a richer range of sequencing notation possible in Little-JIL and hierarchical task network languages such as TAEMS. Thus, Figure 3 prescribes that a task planning instance first retrieves the

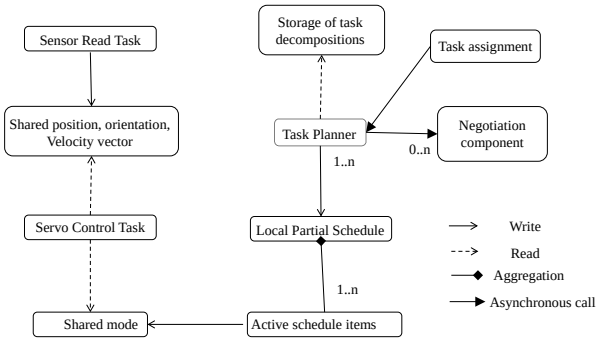

Fig. 2: Architecture

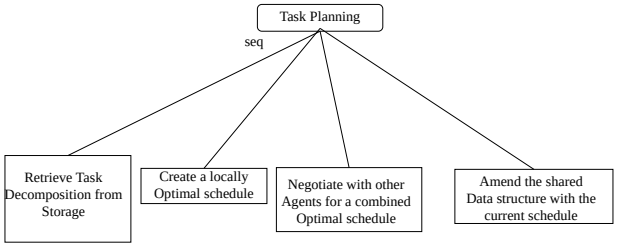

Fig. 3: Task Planning

decomposition for the assigned task, creates a locally optimal schedule (”local” to the individual vehicles that received the assignment), makes the vehicles negotiate to create a combined distributed schedule that reduces the likelihood of redundancies and encourages cooperative behavior. No single vehicle has a global representation of the combined schedule, this is a decentralized solution that trades off how close a combined schedule is to the optimal against the time needed to create such a schedule so that the vehicles could accomplish their tasks by real time deadlines. Once an instance of a planner is finished the local schedule of a vehicle is updated (amended) with schedule items needed to accomplish the task. A local schedule is concurrent itself, it intertwines schedule items needed to accomplish several non-contradicting tasks that a vehicle is assigned. Several instances of planners can concurrently update the local schedule as it is done in blackboard architectures.

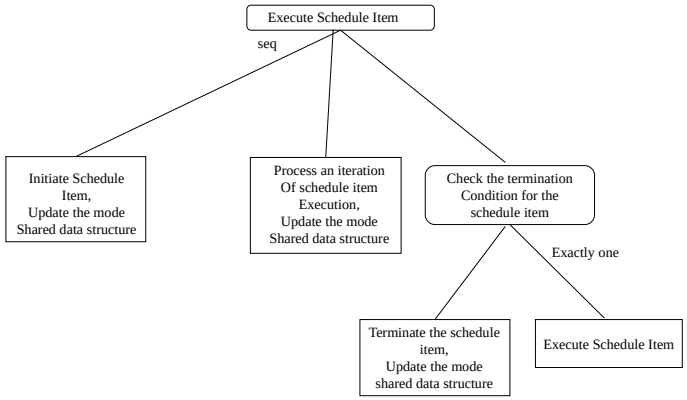

Fig. 4: Schedule Item Execution

Because the local schedule is concurrent, several schedule items can be active at the same time (e.g. flying level to a waypoint and tracking an object with a sensor). An instance of a schedule item can be thought of as a state in a UML statechart. It has an initiation method, a process method that is repeatedly invoked while the state is active (a check for a schedule item completion or abnormal termination can be done in this method), and a termination method. One possible decomposition of a task for a schedule item execution is shown in Figure 4. This definition uses recursion to perform the process method. The currently active schedule items dictate the modes of low level behaviors. For instance, a low level behavior can be a level flight or a turn or a vertical loop or maintaining sensor orientation etc. These are implemented as numerical optimal solutions to the sets of differential equations defining a problem (i.e. a solution is a function in this case). For example, the behavior of maintaining sensor orientation is minimization of the angle between the line of sight to target vector and the sensor orientation vector. The modern state space control theory deals with methods for creation of such numerical solutions (e.g. [9] describes these methods applied to UAV control). Other approaches have been experimented with for this purpose. For instance function approximators such as (evolving) neural networks.

The code that implements these behaviors has to execute at a certain frequency. Actually, it has two big parts: prediction of sensor readings using Kalman filters and response calculations with corresponding servo pulses. The response calculations are the numerical optimal solutions. They are specific to the currently active modes (e.g. level flight to a waypoint optimizing time, or level flight to a waypoint minimizing energy consumption). It is preferable to read sensors at a higher frequency than pulsing servos for the Kalman filters to improve their predictions (say, 100Hz for sensor reads and 50Hz for servo pulses). Thus we are suggesting to run the sensor read code and servo response code in separate threads or in handlers to different hardware timer interrupts (Figures 5 and 6). These two threads can communicate via a shared data structure

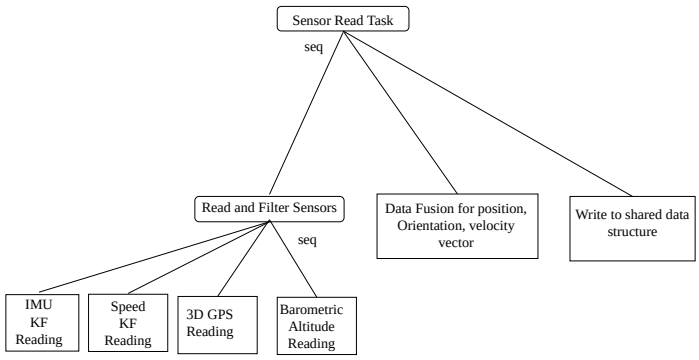

Fig. 5: Sensor Reading Task

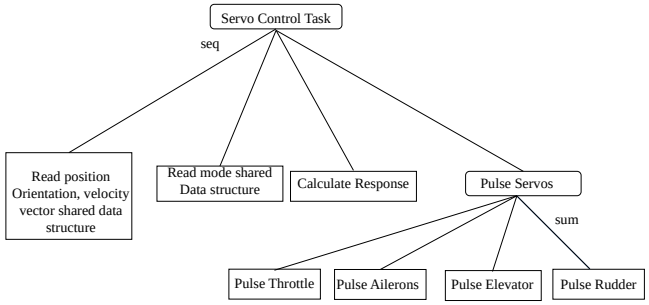

Fig. 6: Servo Control Task

that stores the physical state vector of the vehicle (position, orientation, velocity vector, angular momenta, angle of attack, G-loading). To avoid the possibility of a deadlock or time wasted while blocked we propose to ensure the thread safety of this data structure via the compare and swap operations (accessible in Java via methods on "Atomic" classes). This is where the lack of real time constraints in the Android OS and non-determinism of garbage collection in JVM in our platform of choice introduce certain difficulties. We would like to use light modifications to the Android OS to satisfy the need in real time OS features, for instance as in [3]. For the time being, a simpler solution is to reduce the number of processes running on the embedded Android platform to a bare minimum of the essential ones and minimize the number of instantiations of classes and de-referencing instances to reduce the frequency of the garbage collector invocations.

Now that we have provided a brief description of the software architecture, we would like to note that mapping of this architecture to cores of the hardware platforms is also important. We need to use techniques to reduce context switching and pay attention to the way threads are assigned to processes in a particular OS. Besides we want to experiment with other concurrent frameworks such as the Actor framework in Scala (another language compiled into JVM bytecode). The Actor framework allows for lightweight threads that drastically reduce context switching. Its effectiveness was shown in the implementation of the MapReduce architecture for the Google's cloud computation servers. We also would like to emphasize that our main goals are to provide a model-driven approach for development of control systems with problem solving components, evaluate concurrent architectures and verifications techniques. We do not intend to compete with advanced algorithmic solutions in the area of distributed AI. The next section will itemize the expected research outcomes along these directions.

V. Expected Research Outcomes

The testbed described above should provide for a wide array of experimentation with quantitative results: performance evaluations of architectures and algorithmic solutions, evaluation of how close their solutions are to the optimal while remaining feasible and practical.

Besides, the testbed can serve as an analyzed system for evaluation of verification methods for thread safety of complex shared data structures - task decompositions and concurrent schedules.

Finally, we plan to suggest verification methods for cyber-physical systems with problem solving components based on distributed planning and negotiation (e.g. as an extension of the method described in [6]).

VI. Preliminary Results

By now we have developed a simulated AI framework based on goal oriented behavior and Generalized Partial Global Planning. We have also implemented a low level control system for a four wheeled rover that runs on an Android device. It uses a IOIO board to communicate with motors and servos. We are in the process of building a low cost UAV on the basis of Bixler 2 electrically powered model plane with an Android phone as an embedded control platform. Our next steps are: transfer the simulated AI framework onto an Android device, validate it and generalize its architecture in a process language of choice (Little-JIL) to allow for model-driven reconfiguration and verification of the bodies of the low level behaviors and task decompositions.

VII. Conclusion

This paper highlighted the importance of creation of generalized software engineering methods for design and verification of cellphone-based control systems. It also itemized difficulties in direct transfer of existing methods for this task. Finally, the paper suggested an approach to evaluate solutions to these difficulties.

REFERENCES

[1] http://babelfish.arc.nasa.gov/trac/jpf.

[2] B. Gati. Open source autopilot for academic research the paparazzi system. In *the American Control Conference*, 2013.

[3] I. Kalkov, D. Franke, J. F. Schommer, and S. Kowalewski. A real-time extension to the android platform. In *10th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES)*, pages 105 – 114. ACM, 2012.

[4] Y. Kim and V. R. Lesser. Improved max-sum algorithm for dcop with n-ary constraints. In *AAMAS*, pages 191–198, 2013.

[5] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. N. Prasad, A. Raja, R. Vincent, P. Xuan, and X. Zhang. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, July 2004.

[6] R. Podorozhny, S. Khurshid, D. Perry, and X. Zhang. Verification of multi-agent negotiations using the alloy analyzer. In *Integrated Formal Methods*, 2007.

[7] S. Risi and K. O. Stanley. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial Life*, 18(4):331–363, 2012.

[8] A. Wise. Little-JIL 1.0 Language Report. Technical report 98-24, Department of Computer Science, The University of Massachusetts at Amherst, 1998.

[9] R. Yanushevsky. *Guidance of Unmanned Aerial Vehicles*. CRC Press, 2011.