Mapping an Application to a Control Architecture: Specification of the Problem

Mieczyslaw M. Kokar[1], Kevin M. Passino[2], Kenneth Baclawski[1]
, and Jeffrey E. Smith[3]

[1] Northeastern University, Boston, Massachusetts, USA
kokar@coe.neu.edu
kenb@ccs.neu.edu
[2] Ohio State University, Columbus, Ohio, USA,
passino@ee.eng.ohio-state.edu
[3] Sanders, A Lockheed Company, Nashua, New Hampshire, USA
jeffrey.e.smith@lmco.com

Abstract. This paper deals with self-adapting software that is structured according to a control theory architecture. Such software contains, in addition to its main function, two components - a Controller and a Quality-of-Service module. We show an example of an application and analyze the mapping of this application onto various control theory-based architectures. The application is a radar-based target tracking system. We show how architectural constraints are propagated through the mapping. We also analyze various architectural solutions with respect to stability and time complexity.

1 Introduction

Recently, it has been recognized that change in requirements over the life time of a piece of software is inevitable. This recognition has resulted in the emergence of such research areas as software architectures, software development frameworks, self-adapting software [2, 4, 7], and others. One of the directions in this research (cf. [3]) has been to follow the control metaphor in software development, i.e., to treat the basic software functionality as a *plant* and then add some *redundancy* to control this plant. The redundant software is called the *controller*. The controller is designed to monitor changes in requirements, determine when the software is not meeting those requirements, and make changes to the software (plant) to make sure that it is continually updated so that the requirements are met.

There are various architectures known in the control community. If one wants to follow the control paradigm of engineering software, one needs to know how to map a specific problem onto one of the control architectures. In this paper we consider an example software system and analyze various control architecture realizations of this functionality. In this process, we show how to rationalize some of the architectural decisions.

2 System and Problem Specification

As a case study we have chosen software that implements a resource scheduler for a multiple target tracking domain. The goal of the system is to track multiple targets, i.e., estimate their states (e.g., position, velocity), with satisfactory precision, given the resources (in this case it is one radar). The task must be performed within given time bounds. This kind of a problem has been captured as the following metaphor:

> *Make resource allocation such that it is good enough and it is made soon enough (cf.* Broad Agency Announcement on Autonomous Negotiating Teams, BAA-99-05, DARPA, November 1998*).*

In order to perform its task, the system must allocate resources so that a measure of accuracy of tracking is high ("good enough"). This accuracy depends on the time when the resource (the radar) is given to a specific target (*revisit time*) and the length of the time interval that the radar is measuring that target (*dwell time*). The tracking and the radar allocation functions must be computed within the time constraints dictated by the revisit time of the radar. Moreover, the time for switching the radar among the targets imposes additional constraints on the system. This constitutes the "soon enough" constraint.

According to the approach we are using to specify the architecture, first, we need to view the system in the environment. This is represented in Figure 1. In this figure, the radar measures targets and sends measurements to the tracker, which in turn, controls the radar. The tracker interacts with the (human) operator.

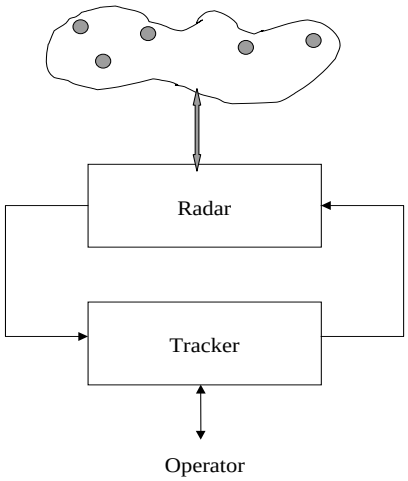

Fig. 1. The Tracking System

2.1 Target Environment Model

The target environment in Figure 1 has N mobile targets (i.e., we assume that the number of targets in the environment is fixed). Suppose that the dynamics of the i^{th} target is given by

$$x^i(k+1) = f^i(x^i(k), u^i(k)) + \Gamma w^i(k) \quad (1)$$
$$y^i(k+1) = g^i(x^i(k+1), u^i(k+1)) + v^i(k+1) \quad (2)$$

for $i = 1, 2, \ldots, N$, where k is the time index and we assume that there is a sampling period of T (hence $x(k)$ represents $x(kT)$, the state at time $t = kT$). Here, $x^i \in \Re^n$ is the state vector of the i^{th} mobile target (e.g., its elements typically contain position and velocity), $u^i \in \Re^p$ is its input (e.g., a steering command), and $y^i \in \Re^m$ is the output (e.g., the output of the radar that senses the position of target i). The nonlinear dynamics are specified via f^i and g^i. These could be developed using first principles of physics to form a continuous time model that is transformed (e.g., using Euler's approximation) to the above discrete time representation. The term $w^i(k)$ in Equation 1 represents "process noise" and Γ describes how this noise is distributed to the states. Typically, $w^i(k)$ is assumed to be zero mean white Gaussian noise with a fixed variance. The "output channel" is given in Equation 2 and there is a "measurement noise" $v^i(k+1)$ given there that represents the lack of accuracy of our sensor (e.g., the inaccuracy of a radar to measure the position of a target). Typically, it is assumed that v^i is zero mean white Gaussian noise with a fixed variance and that v^i and w^i are independent noise sequences. We assume this here also.

The model tells us certain important facts about the environment that impact our ability to track targets. First, note that the number of outputs (measurements) m is typically less than n which represents that we cannot measure the full state of the i^{th} mobile target. Moreover, even if we can measure a component of the state of the target (e.g., position) the measurement is corrupted by the noise v^i, and the problem of estimating the state of the mobile target (which would tell us its position, velocity, and perhaps acceleration) is further complicated by the process noise w^i.

2.2 Tracker Methods and Model

The typical approach to solve the state estimation problem is to use (extended) Kalman filters, one for each of the N targets. In such an approach, the radar is focused on one target at a time so that only one of the N Kalman filters updates its estimate using a measurement, while the others continue estimating the state using only prediction based on the model of the mobile target[1]. It is important to note that, even with the complications mentioned above, the estimate $\hat{x}^i(k)$ of $x^i(k)$ that the Kalman filters produce can typically be made to be reasonably reliable. It is for this reason that in this paper we will assume that

[1] Assuming the measurement-to-track association problem is solved

$$\hat{x}^i(k) \approx x^i(k), k = 0, 1, 2, \ldots$$

for $i = 1, 2, \ldots, N$ so that we can reliably estimate the state of each of the N targets so we know where they are at. We also make this assumption since our focus is not on Kalman filter development, but on the management of the radar to track multiple targets, and in particular on software architectures for how to structure the system that manages the radar.

Besides producing estimates of the state of the mobile targets, the Kalman filters (or other processing algorithms) can provide a characterization of the uncertainty in the estimate (e.g., via the covariance matrix of the innovations process of the Kalman filter). To maintain our focus on the management of the radar, rather than Kalman filtering, we model the uncertainty in the estimate of the state vector of the i^{th} mobile target (when the radar is not focused on the i^{th} target) with

$$\theta_i(t) = p_i t \tag{3}$$

so that we assume that the uncertainty in the state grows linearly with time $t \geq 0$. In discrete time we have $\theta_i(k) = p_i kT$. Later, we will allow p_i to evolve dynamically over time so that the uncertainty in the estimate may be state dependent. For now, we assume that p_i, $i = 1, 2, \ldots, N$, are known constants. We think of p_i as being the rate of uncertainty increase in the information we have about the i^{th} target.

2.3 Scheduler

The task of the scheduler is to manage (point) the radar so as to maintain as much certainty about the locations of the mobile targets as possible. In the above framework it points the radar by picking which of the N Kalman filters in the tracker will be able to use a measurement update to produce its estimate (again, assuming the measurement-to-track association problem is solved). Hence, if it directs the radar at the i^{th} mobile target, it is then directing it so as to reduce uncertainty $\theta_i(t)$ in the estimate for the i^{th} target. Normally, there is a "dwell time" for the radar and, as it is focused on a target for a longer time, it gets more accurate information about that target; however, this is constrained since obtaining perfect information is not possible due to process and sensor noise (and other characteristics). For now, assume that when a radar is focused on the i^{th} target, it reduces the uncertainty about its state estimate at a rate of r_i, $i = 1, 2, \ldots, N$. Hence, pointing the radar at a target for only a very brief time does not result in immediate gain of all possible information (up to the measurement and process noise). Later, we will consider the case where the r_i are not constants, but evolve according to some dynamics.

To summarize how the scheduler interacts with the tracker, note that the model of our process is now given by

$$\begin{aligned}
\theta_1(t) &= p_1 t \\
\theta_2(t) &= p_2 t \\
&\vdots \\
\theta_{i*}(t) &= p_{i*} t - r_{i*} t \\
&\vdots \\
\theta_N(t) &= p_N t
\end{aligned} \tag{4}$$

for the case where the radar is focused on target $i*$ so that it is reducing the uncertainty associated with the estimate of that target's location. It is the task of the scheduler to select $i*$ at each time instant. Inspecting Equation 4 it is clear that the task of the scheduler is to try to maintain bounded values for the θ_i, i.e., it wants

$$\sum_{i=1}^{N} |\theta_i(t)| \leq B \tag{5}$$

for all $i = 1, 2, \ldots, N$ for as small of a value of the bound B as possible (since low values of B provide a characterization of high certainty about the information about the *entire* target environment).

The scheduling problem is complicated by the fact that the radar can only be switched from focusing on one target to another in a finite time so that there is a "delay" $\delta_{i,j}$ in switching from the i^{th} target to the j^{th} target. Clearly, during this delay time the radar is useless and the uncertainties for all the targets goes up. Hence, if the policy for the scheduler switches too often, then it wastes all its time performing the switching. However, if it does not switch often enough, some uncertainty will rise too high. In such a case, during the period of time $\delta_{i,j}$, for all i

$$\theta_i(t) = p_i t \tag{6}$$

There are certain properties of the underlying scheduling problem that are independent of how we specify the scheduler. For instance, there is the concept of the "capacity" of the radar resource. Mathematically, we will not be able to achieve boundedness for Equation 4 if we do not have

$$\sum_{i=1}^{N} \frac{p_i}{r_i} < 1 \tag{7}$$

Additionally, the radar cannot be pointing to more than one direction (target) at a time. Also, the amount of time the radar is pointing to the various targets cannot exceed the time period of one radar sweep, T_r. To express this constraint we need to introduce the variables describing the dwell times for particular targets, τ_i. The constraint then is

$$\sum_{i=1}^{N} \tau_i \leq T_r \tag{8}$$

We will refer to these two inequalities as the "capacity conditions" for the radar resource. It characterizes a necessary condition to be able to achieve boundedness, no matter what scheduling policy we choose.

3 Architectures

This problem can be mapped to various architectures:

1. Centralized One-Module architecture (where one function collects inputs about all the targets and makes radar scheduling decisions as well as computes estimates),
2. Centralized Two-Module (where there are two functions, one for tracking and one for radar scheduling),
3. Decentralized (n functions are allocated to track targets - one function per target - based on the radar measurements, and one function is allocated to schedule the radar)
4. Others, e.g., Decentralized Negotiation (where each of the targets is represented by an agent whose responsibility is to negotiate (with the radar agent) for the radar resource).

The criteria that can guide the architecture selection process are:

1. Overall performance of the system (in this case the level of uncertainty about the states of the targets),
2. Computational complexity (time performance of the system),
3. Controllability (is the goal of the system achievable given the decomposition of the functionality?)
4. Observability (is the state of the system computable given a finite sequence of outputs?)
5. Stability (can the system be organized so that orderly behavior will result?)
6. Scalability (can the system be easily adapted to handle larger amounts of input?)
7. Versatility (includes modularity; relevant for adaptability, i.e., for the ability to adjust to changing situations, e.g., changing number of targets, or changing target dynamics)

In this paper we are focusing only on the issues of stability and computational complexity.

3.1 Stability

A system (e.g., Equation 4) is said to be "stable in the sense of Lagrange" [6] if for every initial condition (e.g., in Equation 4 the $\theta_i(0)$) such that the initial condition lies within a certain bound (e.g., $\sum_{i=1}^{N} \theta_i(0) \leq \alpha$) there exists a bound B such that Equation 5 is satisfied (note that B may depend on α). Clearly, stability in the sense of Lagrange is simply a type of boundedness property. Typically, in practical applications, we would like B to be as small as possible.

Lagrange stability only says that *there exists* a bound for the uncertainty trajectories. A slightly stronger stability condition is that of *uniform ultimate boundedness* (UUB), where for every initial condition the trajectories are bounded, and as time goes to infinity, they will all approach a B-neighborhood of the origin where we know the bound B (e.g., it typically depends on the parameters of the problem - in this case the rates p_i and r_i.

Note that if the scheduler has chosen a specific $i*$ for a long enough time it may be possible to reduce θ_{i*} to near zero. When this happens it normally does not make sense to keep the radar focused on that target (however, if it is a very high priority target then we may want to maintain as much information about it as possible). It is clear that to be able to achieve boundedness, the scheduler cannot ignore any one target for too long (i.e., the "revisit time" for any one target cannot be too long) or its corresponding uncertainty will rise to a high value.

3.2 Complexity

While there are many concepts of computational complexity, the focus of this paper is on time complexity as a function of the number N of targets during a single sampling period. This corresponds to the "soon enough" requirement discussed in Section 2.

In the rest of the paper, we first describe what we mean by "architecture." Then we outline the mapping of our problem to a number of control-based architectures. Finally, we discuss the issues of complexity and stability associated with some of these architectures.

4 What is "architecture"?

The term "architecture" is often used to represent various meanings - from the specification of a structure of the system to be built, through the design of the system, to the global properties of a system. We follow the definition used by the IEEE Working Group on the IEEE P1471 standard [1, 5]. According to this conceptualization, architecture is defined by an *architectural description*. An architectural description must conform to a number of requirements, e.g., it must identify the *stakeholders* and their *concerns*. An architectural description consists of a number of *views* developed according to *viewpoints*. The IEEE P1471 does not specify any specific views, leaving it up to the architect. But it requires

that if there are multiple views, they must identify *inter-view consistency*, i.e., it has to identify consistency constraints and, possibly, any inconsistencies, if they exist. Concerns are specified in *models*. Models can use various representation languages.

In this paper we focus on two views: *user's view* and *structural view*. The user's view captures the requirements on the external interactions between the system under development and its environment. Among others, it captures the user's concern, which in this particular case is the accuracy ("good enough") and the timeliness ("soon enough") of the tracking function. The structural view should include at least three parts: *components, relations* and *consistency constraints*. Accordingly, to map a specific solution to a control architecture, we need to define these three parts.

4.1 System Architecture

First, we specify the architecture of a tracking system without a control loop. Such a system does not have any built-in mechanisms for compensating for disturbances.

We specify this architecture as consisting of three components - the tracker, the radar and the operator, with relations as shown in Figure 1. The constraint is that the tracker knows the states of the targets "good enough", as specified by Equation 5. To specify that the tracker knows the states of the targets "soon enough", we must introduce one more variable, T_c, representing the computation time of the algorithm (i.e., the time needed to perform both the computation of the state estimates and the schedule) and the time to switch the radar from one target to another. In the rest of the paper we do not take into consideration the radar switch time, i.e., we assume that $\delta_{i,j} = 0$. The constraint is then represented by the inequality

$$T_c < T_r \tag{9}$$

It states that the length of the computation is bounded by the radar sweep period T_r. Note that this constraint makes a direct connection to the complexity of the computation. Two more constraints that participate in the specification of this architecture are given by Equations 7 and 8.

4.2 Feedback Based Control Architecture

The Closed Loop (Feedback) Model presented in [3] is shown in Figure 2. We present this architecture here simply to show that, in the following subsections, we will be using this structure as a recurring pattern for various architectural solutions.

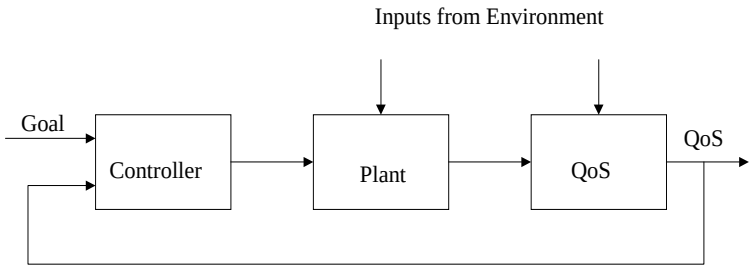

Fig. 2. Conventional Feedback Control Architecture

5 Mapping to Feedback Control Architectures

5.1 One-Module Feedback Based Control Architecture for Tracker

The resulting control-based architecture would include the plant, consisting of two main functions (*track* and *schedule*), the *QoS* module, the *controller* and the *feedback loop*, as shown in Figure 3. We have already discussed the *track*, *schedule* and *QoS* functions. The *controller* for this system would be useful when the parameters p_i, r_i are unknown and thus need to be estimated. Using the control paradigm we would then estimate these parameters incrementally with the amount of increment in each iteration controlled by the *controller* module.

The *Goal* (also called *set point* in the control literature) is expressed here as a value of the constant B (see Equation 5). The constraints are:

$$QoS = \Theta \leq B \tag{10}$$

$$T_c = \sum_{s=1}^{3} T_s \leq T_r \tag{11}$$

where s enumerates computational modules. Equation 11 states the "soon enough" constraint, i.e., that the computation time for all the three modules does not add up to more than the allowed time bound T_r.

5.2 Two-Module Architecture

The basic system architecture, as specified above, involves two functions, *track* and *schedule*, both being part of the same plant. This mapping is not natural

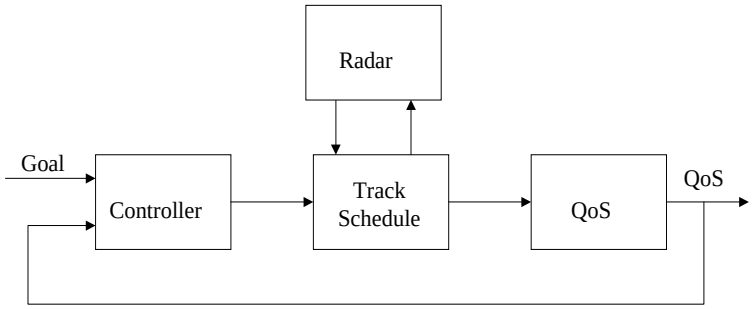

Fig. 3. One-Module Tracker Control Architecture

from the point of view of control theory. In control theory, the plant is considered as a black box that takes inputs and generates outputs. In other words, the plant should be represented by only one function. In order to satisfy this requirement, the plant would need to be split into two, one for *track* and one for *schedule*, as shown if Figure 4.

The control-based architecture would add two controllers, one for each of the two modules, as shown in Figure 5. In general, it also might add two *Goals* and two *QoS* modules. In other words, the overall goal might need to be decomposed. However, for this particular case, we use the same goal, i.e., the goal is still to reduce the uncertainty of the state of the tracked objects. Consequently, we also use the same *QoS* as feedback. The "good enough" constraint would need to bound the computation of all five modules by the allowed time T_c. The constraints are:

$$QoS = \Theta \leq B \tag{12}$$

$$T_c = \sum_{s=1}^{5} T_s \leq T_r \tag{13}$$

Equation 12 represents the "good enough" constraint and 13 represents the "soon enough" constraint.

5.3 Decentralized Architecture

Instead of having one function to track all the targets, we can have a copy of the tracking function instantiated for each target. The system architecture is shown

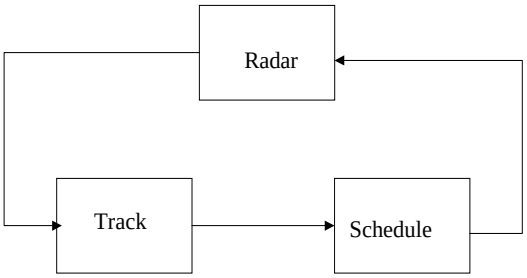

Fig. 4. First Decomposition of Tracker

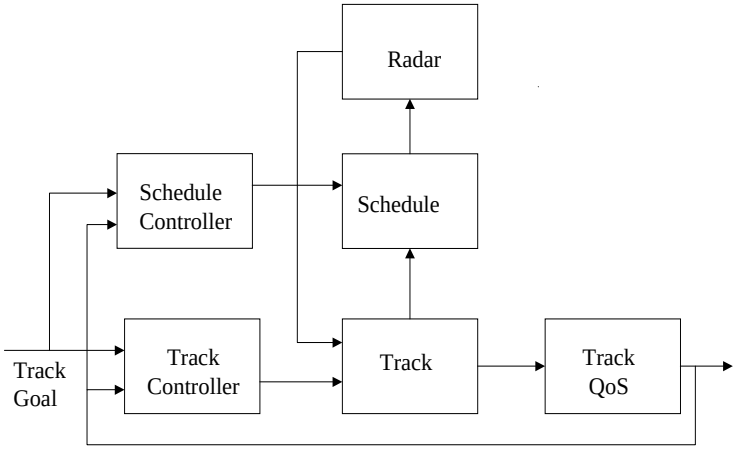

Fig. 5. Control Architecture for the First Decomposition

in Figure 6. This architecture can be mapped onto the control architecture by adding a controller, a QoS module and a feedback loop for each of the trackers and for the scheduler. Additionally, we would need to define *Goal* for both the schedule controller and for each of the tracker's controllers. One possibility is to define bounds B_i on the QoS of each tracker and use them as goals. The goal of the scheduler though must account for all targets. We can use the global bound B in Equation 5 for this purpose. This means that we would need a QoS module to implement this equation. We do not show a drawing of such an architecture since such a figure would be too complicated.

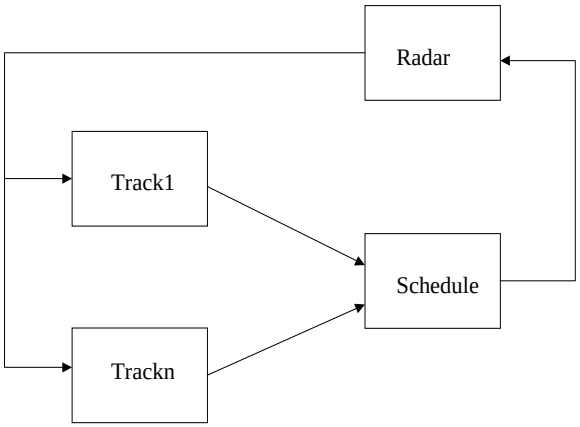

Fig. 6. Decentralized System Architecture

The constraints for this architecture are stated below.

$$QoS_i = \Theta_i \leq B_i,\ i = 1, \ldots, n \tag{14}$$

$$T_c = \sum_{s=1}^{3(n+1)} T_s \leq T_r \tag{15}$$

Additionally, Equation 5 defines the global constraint on the QoS of the scheduler.

6 Comparing Various Architectures

6.1 Stability

Stability is a binary concept; a system can be either stable or not. Typically, we are interested in designs of systems that guarantee stability. Moreover, we may design a system in such a way that it operates in a region that is bounded by a "safety zone" from an unstable behavior. We made such an assumption in the problem specification (see Equation 5). Moreover, we assumed that the problem specification constraints should be preserved in the target control theory based architectures. This assumption is represented as a mapping rule

$$\text{refined\_constraints} \Longrightarrow \text{problem\_constraints} \tag{16}$$

Following this rule, we performed the mapping of the problem onto various control theory based architectures so that the resulting architectures guarantee stability.

As a consequence of this assumption, the stability constraint is not a discriminant characteristic that could be used for comparing the architectures. However, when performing this mapping, we realized that we can satisfy the stability constraint in many different ways. In other words, we could use different types of constraints (Equations 10 through 15), all of which would satisfy the rule given by Equation 16.

The architectural constraints, however, induce a different classification. Note that they constrain the algorithms with respect to both the accuracy of tracking and the time of computation. For instance, in the one-module architecture (where there is no explicit partition between tracking and scheduling), the system designer has a lot of flexibility in how to satisfy the "soon enough" architectural constraint. The designer is (theoretically) free to choose either a shorter computation for tracking or for scheduling, as long as their sum does not exceed T_r. Or the designer may choose an algorithm that would make a decision on where to spend more time (in tracking or scheduling) at run time, as long as the overall computation time is within the required limits. In the two-module architecture, on the other hand, since the architectural constraints are fixed for each of the modules by the architect, the designer is more constrained in the design choices.

Similarly, for the "good enough" constraint, the one-module control based architecture requires the satisfaction of the accuracy constraint by one module. In the two-module architecture, on the other hand, both tracking subsystem and the scheduling subsystem must satisfy the same constraint, and thus the designer has less flexibility in the solution space. The distributed architecture imposes n constraints, i.e., one for each target. This architecture is thus even less flexible from this point of view. Trade-offs between time spent on computation for particular targets must be made under any architecture, however in the one-module architecture the trade-off may be made at the design or the run time, while in the distributed architecture (at least in the approach described in this paper) the trade-off is made at the architecture design time.

6.2 Complexity

The computational tasks involve tracking, scheduling, QoS and control. The complexity of the computation depends on many parameters, such as the precise algorithms used for control and tracking. In this discussion, we focus on the how the complexity increases as the number of targets (N) increases, i.e., how he system scales up.

Consider first the simplest control architecture in which there is just one QoS module and one control module. The QoS and control functions are performed independently for each target, so the time complexity of these tasks are both $O(N)$. Furthermore, this independence property implies that one can perform these two computational tasks on a distributed system. In particular, if there are $\Omega(N)$ processors available, then the time complexity of these two tasks can be reduced to $O(1)$.

The tracking function uses a processing algorithm such as a Kalman filter, as noted in Section 2.2, for each target. Unlike the QoS and control functions, the targets are not necessarily independent. When two targets visually coincide or nearly coincide, it is necessary to disambiguate the two signals. Except for this situation, the time complexity of tracking is $O(N)$. If n targets nearly coincide with one another, the time complexity of disambiguation is $O(n^2)$ for these n targets. Under mild assumptions, such as no more than $\sqrt{N}$ targets may come close to coinciding, the time complexity of tracking remains $O(N)$. As with the QoS and control functions, the basic tracking computation may be distributed. In addition, groups of nearly coincident targets need to be disambiguated, which can also be distributed as long as different groups are not too close.

The last function to consider is the scheduling function. This function is responsible for allocating the radar resource to the targets. Scheduling involves interactions between targets so it cannot be done for each target in parallel. Although resource allocation problems are typically NP-complete, the assumptions embodied in Equation 4 and the assumption that $\delta_{i,j}$ are all equal to 0 imply that this particular resource allocation problem is tractable.

The simplest scheduling algorithm is the one that simply chooses the target with the largest uncertainty, with the dwell time equal to the maximum period of time during which it is possible to reduce the uncertainty for this target. This function can be performed by computing a maximum. The time complexity for this computation is $O(N)$. Unlike the other functions, this computation cannot be performed in time $O(1)$ on a distributed system. By decomposing the computation hierarchically, it can be performed in time $O(\log(N))$ by $\Omega(N)$ processors.

Taken together, the total time complexity is $O(N)$ on a single processor, and the total time complexity is $O(\log(N))$ on $\Omega(N)$ processors. This is the time complexity to compute a single dwell on a single target. For this to be "soon enough", it must not exceed the current dwell time.

More complex scheduling algorithms perform some amount of advance planning to improve performance. This is especially important when the $\delta_{i,j}$ are not zero. The simple scheduling algorithm above will still work when $\delta_{i,j}$ is nonzero,

and one can bound its performance. However, the simple scheduling algorithm is not optimal. In fact, computing the optimum is equivalent to the traveling salesman problem, so it is an NP-complete problem. Nevertheless, small amounts of planning can improve performance without requiring exponential time complexity.

Advance planning takes place during a longer period of time than a single dwell, and it can be as large as the radar sweep time, T_r. In this case, the total complexity T_c must not exceed T_r (Equation 9). Using the simple scheduling algorithm above, the time complexity of scheduling for a radar sweep is $O(N^2)$ on a single processor, and $O(N \log(N))$ when distributed on $\Omega(N)$ processors. Therefore, the "soon enough" constraint for this algorithm is that the time complexity not exceed T_r.

7 Conclusions

In this paper we discussed the problem of mapping an application to a control theory-based architecture. The mapping process consisted of two main steps - first, developing a *basic architecture* of the system without any control loop, and then mapping such an architecture to a control-based architecture. The mapping involves adding a controller and a QoS module to each component of the basic architecture, specification of control goals for each of the controllers, and specification of constraints.

As our case study we have chosen one kind of control theory-based architecture – the feedback control architecture. We considered three basic architectures: one-module, two-module, and distributed ($n + 1$-module).

Often the selection of an architecture is considered as a selection of the structure of the system, i.e., showing a data flow diagram of the system. Clearly, as is shown in this paper, this is not sufficient. The mapping must also include the propagation of the constraints from the basic system to the control theory-based system. This step may be achieved in many different ways. The requirement that we followed in our process was that the satisfaction of the constraints in the control-based system must imply the satisfaction of the system level architectural constraints. It seems natural that the architect should try not to overconstrain the system since such a system would have a narrower domain of applicability (less versatility). Also, the selection of the constraints impacts the complexity of the algorithms that need to be used to implement a given architecture. It is the architect's job to define the constraints wisely so that the impacts are balanced.

In this paper we focused only on two kinds of criteria by which one can compare various architectural solutions: stability and complexity. We formulated constraints in such a way as to guarantee the stability of each of the systems. Therefore, the comparison of architectures by the stability measure does not classify the systems into stable and unstable, since all of them are stable, but into more constrained and less constrained.

Acknowledgments

This research was partially supported by a grant from the Defense Advanced Research Projects Agency.

References

1. R. Hilliard. Using UML for architectural description. *Lecture Notes in Computer Science: Proceedings of UML'99*, pages –, 1999.
2. G. Karsai and J. Sztipanovits. A model-based approch to self-adaptive software. *IEEE Intelligent Systems*, May/June 1999:46–53, 1999.
3. M. M. Kokar, K. Baclawski, and Y. Eracar. Control theory-based foundations of self-controlling software. *IEEE Intelligent Systems*, May/June 1999:37–45, 1999.
4. R. Laddaga. Creating robust software through self-adaptation. *IEEE Intelligent Systems*, May/June 1999:26–29, 1999.
5. Institute of Electrical and Electronics Engineers. Draft recommended practice for architectural description: Ieee p1471/d5.2. Piscataway, NJ, December 1999.
6. K. M. Passino and K. L. Burgess. *Stability Analysis of Discrete Event Systems*. John Wiley, 1998.
7. P. Robertson and J. M. Brady. Adaptive image analysis for aerial surveillance. *IEEE Intelligent Systems*, May/June 1999:30–36, 1999.