Dealing with Emergent Properties in Embedded Systems

Iain Bate

Department of Computer Science, University of York,
York, YO10 5DD, United Kingdom
e-mail: iain.bate@cs.york.ac.uk

Abstract

The development of many systems suffer from unexpected problems during development leading to time and cost penalties. A common situation that leads to problems is when components are integrated and un-anticipated features are formed. These features are often termed emergent properties. In this paper, a compositional approach is adopted where components are parameterised with extra information that can then be used to understand the implications of integration and mapping.

1 Introduction

Emergent properties are an important issue in the development of systems especially those where non-functional properties are considered important or have to be guaranteed. Their importance is due to the fact they can cause anomalous behaviour and hence costly changes [5]. There are many different definitions of the term emergent property, however during the course of this work it is taken to mean those properties that are difficult to predict as they are a by-product of integration and of mapping a design onto a target platform. That is, those properties that could not easily be predicted by simply considering the model before it is mapped onto a platform. The term emergent properties will still be used to describe the properties even though a principal aim of the work is to ease the problems of their prediction so they are no longer classed as emergent. A model is considered to be any appropriate description of a system. In the context of this work no restrictions are assumed concerning the level of abstraction of the model or how the model is represented. The description can be held in a wide range of formats including MATLAB, Ada, assembly language etc..

There are a number of possible strategies for dealing with emergent properties caused by integrating components / models and mapping them onto the target platform. The most popular of these, at least from industry's perspective, is to produce models based on un-realistic models (e.g. assuming static latencies for control systems), test the system and fine tune the model to compensate. The problems with this method are each iteration is expensive in terms of time and money, and often the problems are difficult to overcome leading to significant re-designs.

Another method of dealing with emergent properties is to embed realistic computational models into the overall application model in the first place so that fewer surprises are found when mapping the application. Some of our previous work has taken this strategy [2, 3]. Potential problems with this include, issues can still arise if the reference model is incorrect and the approach is holistic, i.e. it deals with whole systems. The holistic nature of the approach means that if problems do arise then it might be difficult to diagnose and fix the problem as the symptoms may be known but not necessarily the root cause.

The purpose of this work is to investigate an alternative method of solving the problem based on parameterised components (defined here as a replaceable unit which provides a clear and specific function, and featuring a well-defined interfaces) and composition. The basis of the approach is that the information about how emergent properties affect behaviour is embedded into the system at the lowest level and then the components are combined to produce the overall effect. Taking this approach allows the non-functional properties to be established within the model rather than emerge in the later parts of the design. There are two principal benefits of this strategy. Firstly, a detailed understanding of where and how the emergent properties affect the system is available which is beneficial from the perspectives of design, change control and maintenance. Secondly, as the components are composed into larger units, then interim checks can be performed on whether constraints are met. The approach presented in this paper is considered general purpose but is explained using a control systems example.

2 Background on Control Systems

All scheduling approaches require a minimum set of information about timing requirements so that an appropriate scheduler can be produced. For most scheduling approaches the minimum set of information is the deadline and period of tasks [6]. This section explains why these requirements are important in the context of a PID (Proportional, Integral, Derivative) loops.

The main purpose of a PID loop is to ensure the response to inputs is sufficiently fast whilst maintaining the stability, accuracy and limits on data. Figure 1 depicts a typical PID loop used to control the operation of a plant as part of a control system. The control system input is the difference between the input demand (denoted by I), which is the desired plant state, and the plant's actual output (denoted by OP) and it is referred to as the error, (denoted by E). In addition, “real” systems have errors through effects such as measurement disturbance, load disturbance and plant error which are also represented in Figure 1. The PID loop features three gain factors (K_P, K_I, K_D) which are for the proportional, integration and differential parts of the calculation respectively.

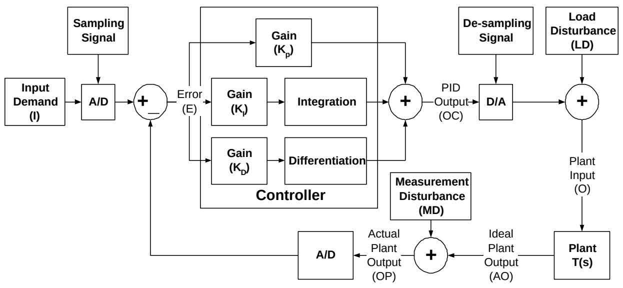

Figure 1. Control System Using a PID loop

Equations (1)-(5) represent the stages of calculation and transformation (in discrete form) performed in the calculations performed by the “Controller” in Figure 1. The equations are formed based on the assumption of a constant sampling rate and latency. In a computer-based control system, these assumptions are unlikely to be valid. Other work, e.g. [2, 3, 6], has explained how variances from the assumptions can affect the overall control systems performance, e.g. introduce errors, affect stability etc., by the use of appropriate timing requirements, i.e. periods and deadlines. In this work knowledge of these effects are built up using parameterised components so that the design reflects reality and hence less prone to unexpected errors.

$$\begin{aligned} E(z) &= I(z) - OP(z) && (1)\\ C_{KP} &= K_P^D \cdot E(z) && (2)\\ C_{KI} &= K_I^D \cdot \sum_{\forall z} E(z) && (3)\\ C_{KD} &= K_D^D \cdot (E(z) - E(z-1)) && (4)\\ OC(z) &= C_{KP} + C_{KI} + C_{KD} && (5) \end{aligned}$$

where $E(z)$ is the current sample of signal E(t)
$E(z-1)$ is the previous sample of signal E(t)

3 Building Emergent Properties into Components

The emergent properties to be considered in this work are value errors and timing. Others including power (consumption and dissipation), memory usage and dependability are not considered for space reasons. From a component perspective, the components needed for a computer-based PID loop are sampling, de-sampling and the calculation step. The following sub-section take each of these components in turn and introduce the emergent properties.

3.1 Sampling Component

The purpose of a sampling component is to convert a continuous analogue signal, $I(t)$, to a discrete digital signal, $I(z)$. In an ideal world, the values of $I(t)$ and $I(z)$ would be the same. However value errors and timing have two main impacts. Firstly converting from an analogue to a digital signal introduces quantisation errors due to the finite resolution available. The error due to the quantisation effect is shown in equation (6). The maximum quantisation error is shown in equation (7) which represents the difference between two consecutive sampled values. It should be noted that the equations assume a linear convertor that does not saturate.

$$quan\_error = I(t) - \left\lfloor \frac{I(t)}{\frac{range(I(z))}{2^{N_{I(z)}}}} \right\rfloor \cdot \frac{range(I(z))}{2^{N_{I(z)}}} \quad (6)$$

where $range(I(z))$ is the range of values representable in the sampled signal and $N_{I(z)}$ is the number of bits used for sampling, e.g. $N_{I(z)} = 14$ for a 14 bit analogue to digital convertor.

$$quan\_error < \frac{range(I(z))}{2^{N_{I(z)}}} \quad (7)$$

The second impact introduces a delay between a signal being available on the input and it being ready for processing. The potential error due to the sampling effect is shown in equation (8). This equates to the difference between any two consecutive samples. The maximum sampling error is shown in equation (9). This is based on the maximum rate of change of the sampled signal (often referred to as its maximum slew rate) multiplied by the sampling period, ΔT_s. A key point these equations demonstrate is the dependence not only only the design but also the application, e.g. slew rate.

$$samp\_error = I(t) - I\left(t - \left|\frac{t}{\Delta T_s}\right| \Delta T_s\right) \quad (8)$$

$$max\_samp\_error = max\left(\frac{\delta(I(t))}{\delta t}\right) \cdot \Delta T_s \quad (9)$$

where $max(f(t))$ is the maximum of a function $f(t)$

3.2 De-sampling

The purpose of a de-sampling component is to convert a digital signal, $I(z)$, to a continuous analogue signal, $I(t)$. In an ideal world, the values of $I(t)$ and $I(z)$ would be the same. However the process introduces timing errors, similar to the previous sampling errors, due to the finite rate at which de-sampling is performed. The cost of quantisation errors can be included using equations (6) and (7).

The errors due to the de-sampling effect are similar to those for sampling given in the previous section. The equations for de-sampling are given in equations (10) and (11). In a similar fashion to the previous sampling errors, there is a mix of variables between application dependent (i.e. rate of change of the input) and design (i.e. the de-sampling rate). It should be noted that it is common for the de-sampling rate to be equal to the sampling rate.

$$desamp\_error = I(z) - I(z-1) \quad (10)$$

where $I(z)$ is the current signal de-sampled, and $I(z-1)$ is the previous signal de-sampled.

$$max\_desamp\_error = max\left(\frac{\delta(I(z))}{\delta z}\right) \cdot \Delta T_d \quad (11)$$

where $max(f(t))$ is the maximum of any given function $f(t)$, and T_d is the de-sampling rate.

3.3 Signal Transformation

For the purposes of this work, it is assumed that the calculation performed does not introduce further significant value errors. As most reasonably priced analogue to digital convertors have less than 16 bits and most reasonably priced microprocessors support 32 bit mathematical operations, even if the transformations are performed in multiple stages it is considered a reasonable assumption. However if further value errors were significant then equations of the form of (6) and (7) could be used.

The signal transformation does however introduce significant latencies. On microprocessor-based solutions the latencies are normally variable. The error due to latency is given in equation (12). The approximate form is then calculated in equation (13) which is based on an instantaneous value for the rate of change multiplied by the latency. Then, the maximum value is given in equation (14) which uses the approximation (equation (12)) in the worst-case scenario where both the rate of change and the latency are maximum. The minimum value for the error due to latency is also given in equation (15) for completeness purposes.

$$\begin{aligned}
latency\_error &= I(t) - I(t - latency) & (12)\\
latency\_error &\approx \frac{d(I(t))}{dt} \cdot latency & (13)\\
max\_latency\_error &\approx max\left(\frac{\delta(I(t))}{\delta t}\right) \cdot R_i & (14)\\
min\_latency\_error &\approx min\left(\frac{\delta(I(t))}{\delta t}\right) \cdot B_i & (15)
\end{aligned}$$

where R_i is the worst-case response time (or latency) of the unit of computation being considered (i.e. a task or transaction), and B_i is the best-case response time (or latency)

Again, the maximum rate of change of the signal is application dependent and the worst-case response time is a by-product of design choices made.

4 Combining Components and Properties

There are two principal methods of combining components. The first method is to treat each component individually, derive its worst-case properties (e.g. error) and then combine using an appropriate method. This approach was taken by Menard [4] where it was shown how noise associated with components could be combined using square functions. The advantage of this approach is its simplicity and scalability, however significant pessimism may result as all related components may not have their worst-case properties simultaneously. Instead in this paper, the combination is to be done mathematically so that a resulting expression is obtained for the whole system. This approach is more precise and with modern tools such as MATLAB and Mathematica then scalability should not be a significant issue. The other reason for choosing this approach is its novelty.

Based on this strategy, in this section the equations (1)-(5) without emergent properties are modified to account for the appropriate errors introduced by the emergent properties identified in section 3. The first equation to be modified is equation (1) which is to be modified to account for emergent property effects on the variables $I(z)$ and $OP(z)$. Equation (16) shows how the variable $I(z)$ provides a modified input variable, $I'(z)$ to account for a sampling rate of T_{si} and quantisation errors caused by the $N_{I(z)}$ bit sampling over the range magnitude $range(I(z))$ and the subsequent calculations. An assumption made throughout this work is that variables neither underflow or overflow. A specific instance of the calculation is given in equation (17) which allows for the maximum sample error between two samples z and $z-1$.

$$\begin{aligned}
I'(z) &= I(z) + max\left(\frac{\delta(I(t))}{\delta t}\right) \cdot \Delta T_{si}\\
&\pm \left(\frac{range(I(z))}{2^{N_{I(z)}}}\right) & (16)\\
I'(z) &= I(z) + (I(z) - I(z-1))\\
&\pm \left(\frac{range(I(z))}{2^{N_{I(z)}}}\right) & (17)
\end{aligned}$$

Similarly, equation (18) shows how the variable $OP(z)$ provides a modified input variable, $OP'(z)$ to account for a sampling rate of T_{so} and quantisation errors caused by the $N_{OP(z)}$ bit de-sampling over the range magnitude $range(OP(z))$. Again, a specific instance of the calculation is given in equation (19) which allows for the maximum sample error between two samples z and $z-1$.

$$\begin{aligned}
OP'(z) &= OP(z) + max\left(\frac{\delta(OP(t))}{\delta t}\right) \cdot \Delta T_{so}\\
&\pm \left(\frac{range(OP(z))}{2^{N_{OP(z)}}}\right) & (18)\\
OP'(z) &= OP(z) + (OP(z) - OP(z-1))\\
&\pm \left(\frac{range(OP(z))}{2^{N_{OP(z)}}}\right) & (19)
\end{aligned}$$

Upon first inspection of equations (17) and (19), it would seem that the variables related to period have been eliminated. However this is not the case as they still feature as part of the time between any two samples, e.g. the time between samples $I(z)$ and $I(z-1)$.

The modified version of equation (1) is given in equation (20) which has been updated to reflect the new variables given in equations (16) and (18).

$$E'(z) = I'(z) - OP'(z) \quad (20)$$

The modified version of variable $OC(z)$, which has been updated to account for the latency that can arise in the computer-based calculations performed is calculated using equation (21).

$$OC'(t) = OC(t) + \left(\frac{\delta(OC(t))}{\delta t}\right) \cdot latency \tag{21}$$

Finally, the variable $OC'(t)$ is then modified to account for the de-sampling rate T_d to give $OC''(t)$ as shown in equation (22).

$$OC''(t) = OC'(t) + max\left(\frac{\delta(OC'(t))}{\delta t}\right) \cdot \Delta T_d \tag{22}$$

A key issue here is that depending on how the system is designed it could be argued that variable $OC''(t)$ shown in equation (22) features a double hit as the error features timing components for both latency and update rate. This version of the equation assumes that the sampling and de-sampling of the system is performed asynchronously from that of the calculations. Therefore the worst-case delay from sampling to de-sampling is $T_{si} + latency$. If a synchronous approach was used, then the equation in (21) with an appropriate latency would be used to calculate $OC''(t)$. A key benefit of having parameterised components and dealing with emergent properties in the way presented, is that the different equations can be used to understand the implications of different design and integration strategies as part of the overall design trade-off analysis associated with a system's development.

5 Use of the Parameterised Components

The approach is demonstrated using the same ball and beam example [1] as in our earlier work [2, 3]. The plant (i.e. the ball and beam) has a variable to be controlled is the position of a free-rolling ball on a beam. It can be mathematically represented by a second-order system using the linearised equation (23). The controller output represents the angle of the gear, $\theta(s)$, and the variable under control is the position of the ball, $r(s)$. The ball and beam is unstable without control and as such is more difficult than examples where the control system is inherently stable.

$$\frac{r(s)}{\theta(s)} = \frac{0.21}{s} \tag{23}$$

The PID equations (1) - (5) were modified to include emergent properties based on equations (16) - (22) and the application specific details from equation (23). Equations (24)-(29) are the result.

The equations show that emergent properties come from four sources. Firstly, properties that emerge from components combined behaviour, e.g. delays due to sequential nature of the data flow between tasks. Secondly, properties that emerge from the mapping of the tasks onto the platform, e.g. errors due to the limited precision of the platform. Thirdly, properties that emerge from executing on the platform, e.g. errors due to latency. Finally, properties that emerge due to "real" world effects, e.g. errors due to load disturbance.

In the full paper these equations were used to derive constraints on the emergent properties in order to meet key control objectives such as maximum output error. This was achieved by feeding in known (worst-case) signals and determining the error. Where appropriate this was maximised using differentiation-based techniques. The results of the evaluation are valid periods and deadlines for which the error was within the tolerable bounds.

$$\begin{aligned}
O(t) &= OC(t) + LD(t) && (24)\\
AO(s) &= T(s) \cdot O(s) && (25)\\
AO(s) &= \left(\frac{0.21}{s}\right) \cdot O(s) && (26)\\
OP(t) &= AO(t) + MD(t) && (27)\\
E(z) &= I(z) + (I(z) - I(z-1)) \\
&\pm \left(\frac{range(I(z))}{2^{N_{I(z)}}}\right) - OP(z) + (OP(z) \\
&- OP(z-1)) \pm \left(\frac{range(OP(z))}{2^{N_{OP(z)}}}\right) && (28)\\
OC(t) &= C_{KP} + C_{KI} + C_{KD} \\
&+ [(OC(t) - OC(t - \Delta T)) \cdot latency] && (29)
\end{aligned}$$

6 Conclusions

In this paper, a means of dealing with emergent properties, e.g. latency, resolution error etc, has been presented. The aim of the work has been to embed information into components (e.g. resolution and latency) that can then be used during the integration of components and subsequent mapping onto a given platform in order to reduce the likelihood of emergent properties. The work has then been demonstrated using a control systems example but full details are not included for space reasons.

References

[1] www.engin.umich.edu/group/ctm/examples/ball/ball.html.

[2] I. Bate, A. Cervin, and P. Nightingale. Establishing timing requirements and control attributes for control loops in real-time systems. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 121–128, 2003.

[3] I. Bate, J. McDermid, and P. Nightingale. Establishing timing requirements for control loops in real-time systems. *Journal of Microprocessors and Microsystems*, 27(4):159–169, 2003.

[4] D. Menard and O. Sentieys. Automatic evaluation of the accuracy of fixed-point algorithms. In *Proceedings of DATE*, pages 529–537, 2002.

[5] E. Rechtin. The synthesis of complex systems. *IEEE Spectrum*, 34(7):50–55, 1997.

[6] M. Torngren. Fundamentals of implementing real-time control applications in distributed computer systems. *Real-Time Systems*, 14(3):219–250, May 1998.