View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Loughborough University Institutional Repository

Process and Tool Support for Real-Time Performance Analysis of Integrated Modular Systems

A. Grigg[1], L. Guan[2], P.R.Baalham[1], S.G.Manuel[2]

[1] Systems Engineering Innovation Centre (SEIC), Loughborough, UK
(a.grigg@lboro.ac.uk, p.r.baalham@lboro.ac.uk)
[2] Dept. of Computer Science, Loughborough University
(l.guan@lboro.ac.uk, s.g.manuel@lboro.ac.uk)

Abstract
This paper describes a real-time system performance analysis methodology and toolset that has been developed at SEIC to be an integral part of a broader BAE Systems Military Air Solutions (MAS) process and toolset for Integrated Modular Systems (IMS). The proposed modelling approach and toolset components provide some key 'through-life' real-time system engineering benefits relating to system performance, including : the ability to construct a performance prediction model during the early stages of system design and to independently model the timing behaviour of end-to-end transactions across a distributed system of shared processing and network resources.

Keywords – Real-Time Systems; Performance Analysis; Predictability;. Integrated Modular Systems; Tool Support.

1 Introduction

This paper describes a real-time system performance analysis methodology and supporting toolset that have been developed for *integrated modular systems* (IMS) [1]. The solution is not specific to IMS, however, but generally applicable to a much broader class of real-time distributed systems that require evidence of predictable performance to be generated during development. The ability to demonstrate that the final target system has met its timing requirements is a key capability in the development of a real-time system. This can be carried out by constructing a model of system timing behaviour that can be used to make predictions about maximum response times, communication delays, delay variations and resource utilisation. A performance modelling solution has been developed that provide such capabilities. The approach, known as *reservation-based analysis* (RBA) [2], also provides some further 'through-life' system engineering benefits :

- the ability to construct a performance prediction model during the early stages of system design and then continually evolve this model to incorporate increasing design and implementation details – this allows engineers to make ongoing predictions about system performance throughout development based on the latest information available; in turn, system upgrade scenarios can be similarly evaluated for potential performance properties;
- the ability to independently model the timing behaviour of specific end-to-end transactions across a distributed system of shared processing and network resources and, correspondingly, limit the scope of re-analysis/verification in the event of localised changes to system requirements or implementation details.

The rest of this paper gives an overview of the RBA approach and the recently developed toolset components that support the construction and automated execution of the system performance model.

2 Performance Modelling Motivation and Approach

Timing analysis models are normally developed in a 'bottom-up' manner, ie. the model is not finalised until after the implementation and integration details of the system have been decided. Hence, it is not possible to assess the performance of the system until late in the development process. The results of analysis performed at this late stage of development are, of course, essential to support final verification of the system timing requirements. Deficiencies discovered at this late stage, however, can give rise to significant re-work, the costs of which are typically significant and a major factor in the overall development cost of industrial real-time systems. The risk of re-work associated with the development and verification of the timing properties of the system can be reduced by making the notion of timing analysis more integral to the systems engineering process as a whole, allowing it to be applied throughout the development of the system, starting much earlier in the process. This allows an ongoing assessment of emergent system performance properties relative to specified timing requirements and also provide progressive guidance on the selection of design/implementation details at successive stages of development.

In the earlier stages of development, performance-related information will be scarce for most or all parts of the system. Even if the system level timing requirements are well defined and decomposed to varying extent during design stages, the ability to perform timing analysis relies

fundamentally on the notion of resources – some media through which the logical components of the system design/implementation can be executed. Processing resource and scheduling details are required for calculating worst-case execution and response times of software components. Analogous details of communication media are required to determine worst-case communication delays. In order to perform timing analysis prior to the system implementation stage, it is therefore necessary to work with an implementation-independent, abstract model of system resources. When implementation details are later finalised, performance predictions acquired via the abstract model can then be verified for the target system. This gives rise to a two-stage approach to timing analysis :

- abstract timing analysis – performed during the definition and decomposition stages of development on the basis of the abstract resource model; the net result is a set of worst-case (and best-case) guarantees regarding the timing behaviour of the system that are subject to a set of obligations being met by the final target implementation;
- target-specific timing analysis – performed during the system implementation and integration stages of system development, the aim being to demonstrate that the set of obligations imposed during the abstract timing analysis phase have actually been met.

In RBA, the end-to-end timing properties of the system are captured, decomposed and analysed in terms of real-time *transactions*. The transaction model is hierarchical, in the form of an acyclic, directed, nested graph, capturing an evolving system definition during development. The leaf nodes of the graph capture the concurrent processing and communication elements within the transaction, termed *activities*; non-leaf nodes are referred to as nested transactions. The edges of the graph capture the precedence and nesting relationships within the transaction. The parameters via which timing behaviour is represented and observed are the same for a single activity, a group of related activities, a nested transaction and a system level transaction, thus providing a highly composable and scalable model.

For any given transaction or activity $\lambda_{i,..,k}$ the basic timing parameters are represented in RBA as follows :

- input jitter, $J^{in}_{i,..,k}$ - the maximum width of the time window that spans the arrival of all associated input events;
- output jitter, $J^{out}_{i,..,k}$ - the maximum width of the time window that spans the delivery of all associated output events;
- minimum I/O separation, $d_{i,..,k}$ - the minimum separation in time between input and output events;
- minimum inter-arrival time, $a_{i,..,k}$ - the minimum separation in time between successive input event windows.

These timing parameters are inter-related via the minimum and maximum response times for $\lambda_{i,..,k}$ (denoted by $r_{i,..,k}$ and $R_{i,..,k}$) as shown in Figure 1.

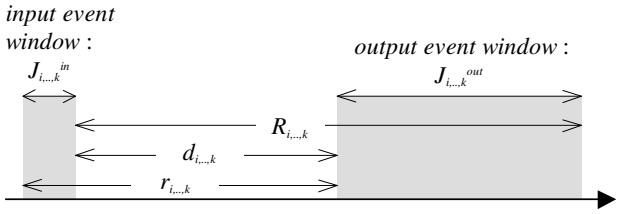

Figure 1 – *Basic Timing Parameters*

For each given transaction topology and assignment of timing parameters, the transaction level end-to-end delays and jitter can be expressed as a function of the leaf-node activity response time parameters. This involves a depth-first traversal of the topology graph, accounting for activity level processing and communication delays, precedence relationships and nesting relationships at each stage – see [3] for details and examples. The approach can be used in either a bottom-up or top-down manner, ie. to determine transaction level delays from activity level response times or, as part of a more structured decompositional approach to achieve performance by design, to derive activity resource usage constraints directly from system/transaction level requirements.

In the latter context, an abstract (target-independent) scheduling model is provided in RBA that can be used as a basis for schedule implementation and analysis. This model links activity level response times and resource usage constraints via a bandwidth reservation model referred to as the *rate-based execution* model. A range of compliant cyclic and priority-based scheduling solutions is then available to implement the final target schedule, guided by the execution space constraints illustrated in Figure 2, where $c_{i,..,k}$ and $C_{i,..,k}$ denote the minimum and maximum resource usage requirements and $v_{i,..,k}$ and $V_{i,..,k}$ denote the minimum and maximum execution rates.

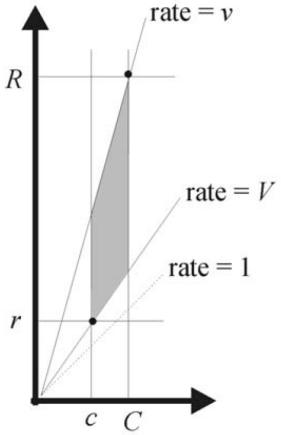

Figure 2 – *Valid Execution Space for Activity* $\lambda_{i,..,k}$

Significantly, from a broader systems engineering perspective, the predictions obtained from the abstract rate-based execution model are target-independent. The final target-dependent predictions are ultimately dependant on the final scheduler implementation but can be guaranteed to be within the bounds predicted from the target-independent model by adopting an RBA rate-based execution model compliant scheduling solution. An example of such a solution is given later in the paper (see section 8).

3 RBA Toolset Structure

The process and toolset allow the RBA mathematical framework and accompanying set of scheduling rules to be applied as part of an evolving avionics system/software design model as follows :

- the system/software design model, expressed in UML, is annotated with performance related information using a defined RBA-UML profile; performance attributes can be based on simple estimates/budgets, actual measurements or analytical predictions (such as from source code analysers); specific end-to-end transactions are then captured within the design model to be the target of analysis;
- an automatic RBA-XML file generation utility is used to export the UML model data from the design tool; the export utility embodies an RBA-XML schema to ensure validity of the exported XML file;
- the XML file is then imported into MATLAB and the analysis is performed to determine system level performance properties including maximum end-to-end delays, maximum delay variations and resource utilisation profiles; numerical and simple visual representation of results are provided to the engineer.

Each of these toolset components and corresponding stages of the performance modelling process are described in the sections that follow.

4 RBA UML Model Construction

The first stage of the process is to create the source RBA UML model in the required form. The source model will typically be constructed within an overall IMS/system design model in order to use pre-defined components from that model as a means of reducing duplicated design effort and achieving consistency. The structure and contents of the non-RBA parts of the IMS/system model are entirely at the discretion of the system designer. The RBA model structure contains three main packages :

- *rba_configurations* package, containing all the RBA model data specific to the IMS/system under design;
- *rba_connectors* package, containing some generic reusable components to aid RBA model construction;
- *rba_profile* package, containing the generic definition of the UML profile for RBA.

Note that the *rba_profile* package and the *rba_connectors* package contain standard model components. The user is free to extend the contents of the *rba_connectors* package but not the *rba_profile* package, which contains the following UML stereotype definitions :

- *rba_activity*, to identify UML model elements as RBA activities, with tag values to capture associated timing properties as follows :
  - *rba_utilisation*, a utilisation/bandwidth budget value in the range [0, 100]%;
  - *rba_resourceId*, a cross-reference into the IMS model to identify the allocated (processing) resource;
  - *rba_jitterTolerance*, a maximum value on permitted activity release jitter (in milliseconds);
  - *rba_activityRef*, a cross-reference into the IMS model to identify the IMS software component associated with this activity;
- *rba_connector*, to create precedence relationships between activities; with a single tag definition as follows :
  - *rba_connectorRef*, a cross-reference into the IMS model to identify the associated IMS 'virtual' communication channel (VC);
- *rba_merge*, to create many-to-one precedence relationships in the construction of end-to-end transactions;
- *rba_trigger*, to create transaction trigger events.

The *rba_configurations* package contains all of the RBA model data that is specific to the IMS/system under design. Each configuration is in fact contained in a separate sub-package to capture the RBA activity and transaction model data specific to each static system configuration (in a similar manner to how the IMS *Blueprint* information is stored separately for each configuration). For each static configuration, a value Δ is specified – in simple terms this equates to the minor cycle time in a cyclic scheduling solution (although the full RBA model is more general and can support a different value of Δ for each activity, corresponding different granularities of bandwidth allocation – a future toolset enhancement waiting to happen). Each RBA transaction that exists (or at least is needed to be modelled) in the configuration is then captured as a UML activity diagram, as illustrated below :

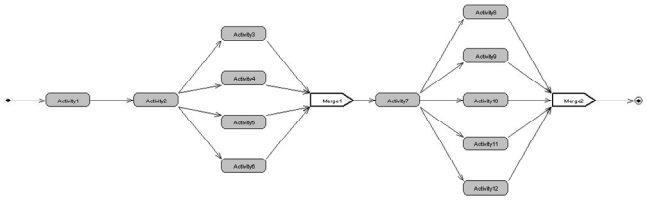

Figure 3 – *Example RBA Transaction as UML Activity Diagram*

The *rba_connectors* package contains a set of generic connector components that can be reused to construct the IMS/system specific model. At present, the package only holds one such component, the merge connector, which can be used in the UML activity diagrams to create many-to-one precedence relationships in RBA transactions.

5 UML Model Data Export to XML

The second stage of the process is to export the RBA UML model data to an XML file format. Current RBA toolset development work is using the Artisan Studio UML tool which offers a variety of ways to export data from the UML model into other formats, including an API that can be accessed via Visual Basic code; a standard XMI file export facility and a Web Publisher facility (the chosen approach for reasons of simplicity of use and completeness of exported data).

The Artisan Studio Web Publisher exports the UML model data to a compound HTML/XML format – a master HTML file that references multiple XML files. A Visual Basic programme has been written to then convert these outputs into the required XML format as specified by an RBA XML schema. The user interface for this toolset component is shown below, illustrating how the user can select an arbitrary subset of configurations and transactions to be the focus of subsequent performance analysis.

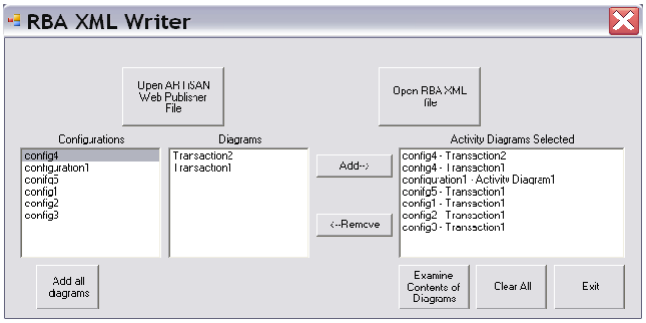

Figure 4 – *RBA XML Writer User Interface*

In subsequent user views, the basic model elements and performance data can be reviewed prior to final XML file generation.

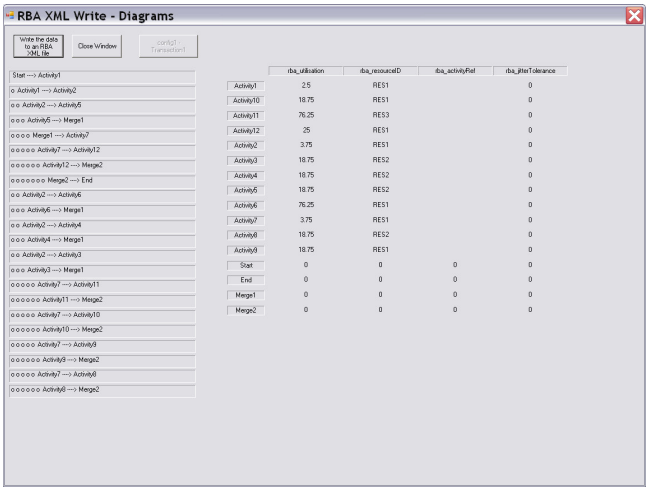

Figure 5 – *Example Transaction Data View*

The RBA XML Writer also allows previously exported XML files to be reimported for review at a later date if required.

6 XML Schema and Instance File

This subsection gives a short description of the XML schema and corresponding instance file format. At the head of the file is a set of basic element declarations and at the end of the file is a set of basic type definitions. The main body of the file is then encapsulated in the top level rba_model element which is comprised of a sequence of rba_configuration elements :

```
<xs:element name="RBA_Model">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="rba_configuration"
    minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

The rba_configuration element is then composed of a sequence of rba_activity and rba_transaction elements :

```
<xs:element name="rba_configuration">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="delta" type="delta"
    maxOccurs="1"/>
   <xs:element ref="n_act"
    maxOccurs="unbounded"/>
   <xs:element ref="rba_activity" minOccurs="1"
    maxOccurs="unbounded"/>
   <xs:element ref="rba_transaction" minOccurs="1"
    maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute ref="name"/>
 </xs:complexType>
</xs:element>
```

Each rba_activity element has a number of associated timing properties corresponding to the UML model rba_activity stereotype tag values :

```
<xs:element name="rba_activity">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="rba_act" maxOccurs="1"/>
   <xs:element ref="rba_u_act" maxOccurs="1"/>
   <xs:element ref="rba_res_list" maxOccurs="1"/>
   <xs:element ref="rba_jitterTolerance"
    maxOccurs="1"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Each rba_transaction element is then defined by a sequence of activity connections (precedence/ordering relationships), each of which is identified by a source and destination :

```
<xs:element name="rba_transaction">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="n_act_trans" maxOccurs="1"/>
   <xs:element ref="connection" minOccurs="1"
    maxOccurs="unbounded"/>
   <xs:element ref="trans_act" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute ref="name"/>
 </xs:complexType>
</xs:element>
<xs:element name="connection">
 <xs:complexType>
  <xs:attribute ref="source"/>
  <xs:attribute ref="destination"/>
 </xs:complexType>
</xs:element>
```

The rest of the XML schema contains the basic type definitions required to support these elements.

7 XML Import into MATLAB

The third stage of the process is to import the XML file data into MATLAB for the analysis to be performed. To this end, an XML-to-MATLAB code generator has been developed. The code generator is structured as follows :

- a front-end that parses the (validated) XML instance file and creates the analogous data structures in the MATLAB workspace;
- a back-end that converts the workspace data into a series of MATLAB M-files, one for each configuration itemised in the source XML file.

Each M-file contains all the global/system level, activity level and transaction level data required to perform the RBA analysis on a single configuration.

8 Performance Analysis in MATLAB

Firstly, each system resource (processor or communication medium) is analysed independently given the following set of information :

- the set of allocated model elements (activities) and their attributes;
- the details of the local resource scheduling (or medium access control) policy;
- the resource requirements of any other software components requiring access to the resource (such as operating system components, middleware services, I/O drivers, interrupt service routines etc.) not explicitly captured in the source model.

The second point above is a significant factor in deciding the form of localised resource level analysis to be used. For the purposes of initial toolset development and demonstration, one form of analysis for cyclic scheduling and two different forms of analysis for static priority-based scheduling (different mathematical models with different degrees of accuracy) have been implemented. This serves to indicate the impact of scheduling policy and choice of analysis method on the performance predictions obtained.

Having obtained the results of the analysis of each system resource with its allocated activities, these results are then fed into an end-to-end transaction model to determine the overall system level response times.

To illustrate these stages of schedule definition and analysis, Table 1 gives an example set of timing attributes imported into MATLAB from the UML model via the XML file – this is a uniprocessor example based on the Generic Avionics Processor (GAP) defined in [3]. Each GAP 'task' is modelled as a single RBA activity since there is no benefit in further decomposition in this example. All GAP tasks are periodic with period $T_j=R_j$, except for τ_{10} which is sporadic with minimum inter-arrival time a_{10}=200. Since no input jitter is specified for the periodic tasks, it is assumed that $a_j=T_j$ for these tasks. Conversely, assigning $a_{10}=T_{10}$ for the sporadic task (the value of 200 shown in brackets in the table) gives a total task set utilisation requirement of 83.5%.

j	*Function*	C_j	R_j	$v_j=U_j$
1	Radar Track Filter	2	25	0.08
2	RWR Contact Mgt.	5	25	0.2
3	Data Bus Poll Device	1	40	0.025
4	Weapon Aiming	3	50	0.06
5	Radar Target Update	5	50	0.1
6	Nav. Update	8	59	0.1355
7	Display Graphic	9	80	0.1125
8	Display Hook Update	2	80	0.025
9	Target Update	5	100	0.05
10	Weapon Protocol	1	200	0.005
11	Nav. Steering Cmds.	3	200	0.015
12	Display Sores Update	1	200	0.005
13	Display Keyset	1	200	0.005
14	Display Stat. Update	3	200	0.015
15	BET E Status Update	1	1000	0.001
16	Nav. Status	1	1000	0.001

Table 1 – *Example Timing Attributes*

8.1 Basic Cyclic Implementation Scheme

The basic cyclic implementation scheme can be applied to derive an RBA-compliant schedule by first selecting an appropriate value for the cycle time Δ. Then, for each activity λ_j :

- the 'normalised' response time $R_j^{\Delta} = \left\lfloor \frac{R_j}{\Delta} \right\rfloor \Delta$;
- the 'normalised' execution rate $v_j^{\Delta} = \frac{C_j}{R_j^{\Delta}}$;
- the time δ_j for which the task must be executed in each cycle Δ $\delta_j = \lceil v_j^{\Delta} \Delta \rceil$;
- the guaranteed response time (target-independent) $R_j^{\delta} = \left\lceil \frac{C_j}{\delta_j} \right\rceil \Delta$;
- the minimum run-time execution rate $v_j^{\delta} = \frac{\delta_j}{\Delta}$;
- the guaranteed computation time $C_j^{\delta} = \frac{R_j^{\Delta}}{\Delta} \delta_j$.

Given the schedule construction constraint $\Delta \le \min_j R_j$, assign Δ=25. This leads to the cyclic scheduling parameter assignment and response time results given in Tables 2, 3 and 4 (and their associated figures).

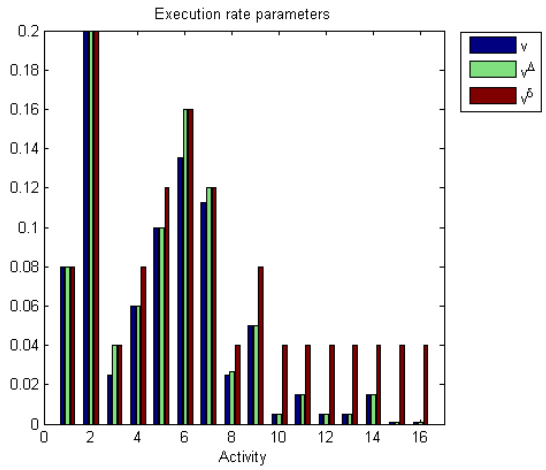

Figure 6 – *Example - Cyclic Schedule Implementation (Execution Rate Parameters)*

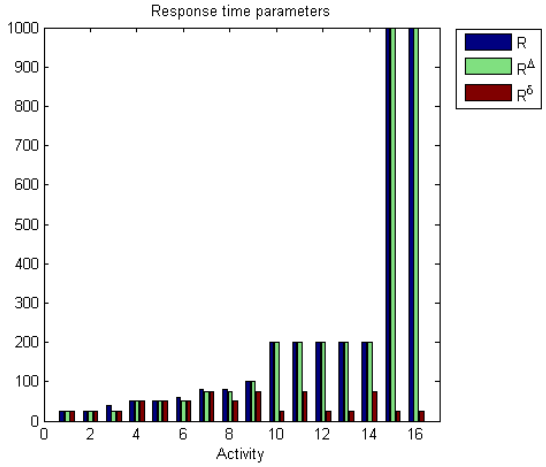

Figure 7 – *Example - Cyclic Schedule Implementation (Response Time Parameters)*

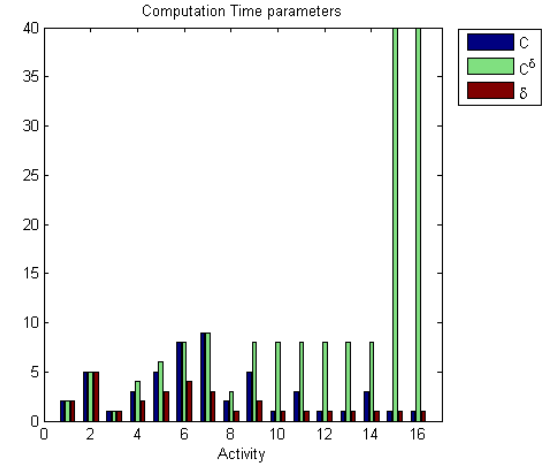

Figure 8 – *Example - Cyclic Schedule Implementation (Computation Time Parameters)*

A number of observations can be made from these results. From Figure 6, the sum of the initial execution rate parameters (v_j) corresponds exactly to the total utilisation requirement of the task set (83.51%). This arises since the worst-case response time of every task is equal to its minimum inter-arrival time. After defining a cycle time of Δ=25, the sum of the rate parameters (v_j^{Δ}) corresponds to a total bandwidth allocation of 88.37%, a noticeable but reasonable increase compared to the true requirement. At the final stage of calculation, however, the need to provide integer values for the final rate parameters (v_j^{δ}) gives rise to a significant over-allocation of bandwidth due to the combination of rounding effects for the overall task set. The final bandwidth allocation is 120% and, hence, the extent of the over-allocation is sufficient to make the task set no longer schedulable on a single processor (by this scheme). The cyclic schedule has been constructed, however, to allow individual activities to be removed (or have their timing attributes changed) without affecting other activities in the schedule. Hence, it is straight forward to reduce the task set to one that is schedulable on a single processor by simply removing one or more activities (to be reallocated to another processor) until the final bandwidth allocation is less than 100%. The ability to manipulate the schedule in this manner is a considerable benefit in the context of engineering larger-scale real-time systems.

A counter effect of bandwidth over-allocation is an equivalent reduction in worst-case response times (R_j^{δ}) compared to the stated requirements (R_j), as can be seen in Figure 7. For example, λ_{15} has a final bandwidth allocation of 4% (equivalent to its execution rate of 0.04) compared to its stated requirement of 0.1%. The corresponding reduction in its worst-case response time is apparent in the final value of 25 compared to an original requirement of 1000. The over-allocation of bandwidth is due to the restriction that every task is executed (for a duration δ_j) in every cycle Δ, as reflected in the final computation times (C_j^{δ}) given in Figure 8. This restriction leads to a simpler (and more readily modifiable) scheduling solution but can be lifted to allow a more flexible scheme to be defined in favour of reducing the bandwidth over-allocation. Such a scheme is described and illustrated below.

Note that the basic scheme does *not* compromise the true timing requirements of the task set – there is no imposition of false iteration rates for the purposes of constructing a schedule (a criticism often levelled at cyclic scheduling solutions). Furthermore, the schedule is incrementally modifiable such that schedulability can be maintained following activities being added, removed or modified by merely ensuring that the final bandwidth allocation is less than 100% (and that the choice of Δ is still suitable).

8.2 Cyclic Server Implementation Scheme

As suggested above, it is possible to reduce the bandwidth over-allocation associated with the basic cyclic implementation scheme by relaxing the constraint that every activity must be offered the chance to execute in every cycle. This gives rise to the cyclic bandwidth server scheme. The starting point is once again the selection of a cycle time Δ subject to the same constraint. Then define a the server activity $\lambda^S(\delta^S,N^S)$ as a notional activity that is allocated δ^S execution time units in every Δ cycle but does

not actually consume that allocation itself. Instead, the server offers the resource to other activities so that these can execute with an effective cycle time of $N^S\Delta$. The total bandwidth of the server can then be used to execute a number of activities that individually have relatively low bandwidth requirements that would otherwise be allocated a disproportionate amount of bandwidth by the basic scheme. The cyclic server solution exploits the fact that the basic scheme does not require the execution time allocated to an activity within a scheduling cycle to be contiguous. The analysis associated with the cyclic server is, therefore, exactly analogous to that for the basic cyclic scheme but with Δ replaced by $N^S\Delta$. The cyclic server method is actually a generalisation of the basic cyclic scheme described above, where multiple cycle times are supported.

The following example illustrates the use of the cyclic server method to improve bandwidth allocation compared to the basic cyclic implementation scheme. Assuming the same basic cycle time Δ=25, define a server $\lambda^S(2,40)$ to execute the low utilisation activities $\{\lambda_{10}, \ldots, \lambda_{16}\}$. Figure 9 shows the improved results under this scheme (the values of other parameters not shown in the table are the same as before under the basic cyclic scheme).

The total capacity of the server $\lambda^S(2,40)$ is $\nu^S = 0.08$. Hence, 92% of the total processor capacity is available for non-server-based activities $\{\lambda_1, \ldots, \lambda_9\}$ and 8% for server-based activities $\{\lambda_{10}, \ldots, \lambda_{16}\}$. So, whilst the total bandwidth allocation is more efficient than for the basic scheme - 97.5% compared to 120%, this is not sufficient to guarantee feasibility on a single processor – it is also necessary to show separately that activities $\{\lambda_1, \ldots, \lambda_9\}$ can be executed within their 92% allocation and that activities $\{\lambda_{10}, \ldots, \lambda_{16}\}$ can be executed within their 8% allocation. From Figure 9, the combined allocation for activities $\{\lambda_1, \ldots, \lambda_9\}$ turns out to be exactly 92% and the combined allocation for activities $\{\lambda_{10}, \ldots, \lambda_{16}\}$ is 5.5%. Hence, the complete set of activities *is* schedulable on a single processor under this scheme. The improved efficiency of this scheme is also reflected in the increased number of activities that have been allocated the exact bandwidth to meet their requirements – 10 out of the 16 activities now, compared to only 4 previously.

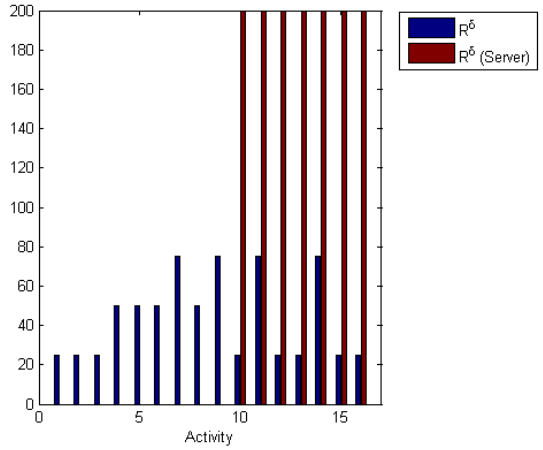

Figure 9 – *Example - Cyclic Server Implementation (Execution Rate Parameters)*

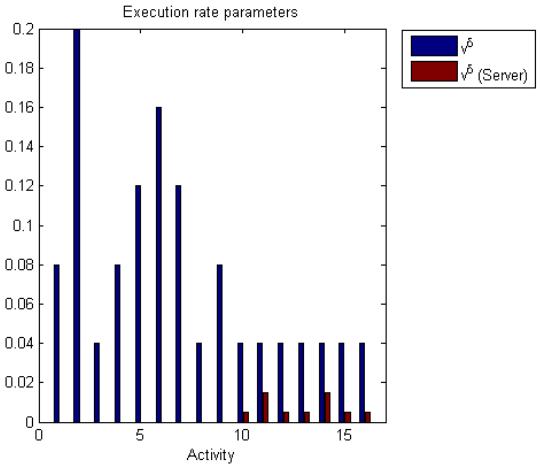

Figure 10 – *Example - Cyclic Server Implementation (Response Time Parameters)*

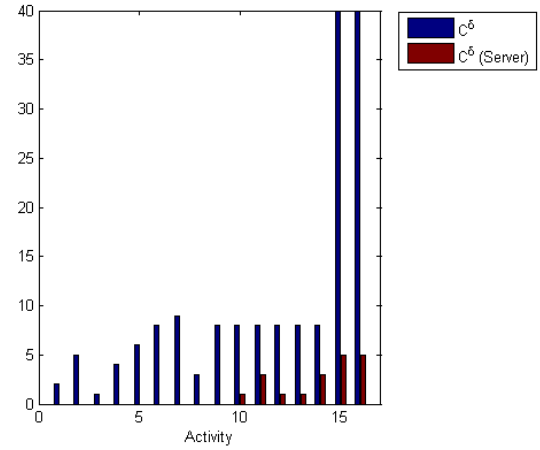

Figure 11 – *Example - Cyclic Server Implementation (Computation Time Parameters)*

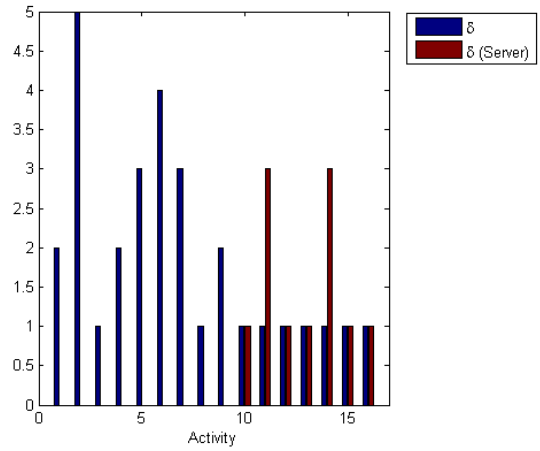

Figure 11b – *Example - Cyclic Server Implementation (Computation Time Parameters)*

Having obtained the results of the analysis of each system resource with its allocated activities, these results are then fed into the end-to-end transaction model to determine the overall system level response times.

9 Conclusions

The RBA performance modelling approach and toolset have been developed to provide through-life performance modelling support to the systems engineering process. The following toolset concepts and components have been implemented so far :

- a UML model structure and profile for RBA that integrates with the BAE Systems MAS IMS model structure and profile;
- an automatic XML file generator from UML model data (currently in Artisan Studio);
- an XML data representation and validation schema;
- an automatic MATLAB scenario file generator from XML model data;
- a suite of schedule definition and analysis routines in MATLAB.

Work continues on the toolset development and technology transfer to BAE Systems MAS.

10 References

[1] J.Kemp, A.S.Wake, B.Williams, "Development of the ASAAC Software Architecture," *Proceedings of ERA Avionics Conference and Exhibition*, 2000.

[2] A.Grigg, "Researvation-Based Timing Analysis – A Partitioned Timing Analysis Model for Distributed Real-Time Systems," *University of York Thesis YCST-2002-10*, 2002.

[3] C.D.Locke, D.R.Vogel, T.J.Mesler, "Building A Predictable Avionics Platform in Ada," *Proceedings of IEEE Real-Time Systems Symposium*, 1991.