Specifying Properties of Distributed Real-time Systems for Dynamic Resource Management

Randy Coleburn (SRC), Rose Daley (John Hopkins University Applied Physics Lab.), B. Dasarathy (Telcordia), Patrick Lardieri (Lockheed Martin), Paul Legault & Brad Orner (Raytheon), Rick Schantz (BBN), Jonathan Tanner (University of Illinois), Lonnie Welch (Ohio University), and Paul Work (Raytheon)

Abstract

This paper discusses the resource management approach being developed in the DARPA ARMS program. The primary focus of the paper is a specification language for the characterization of the software systems and hardware resources to be managed. The approach differs from previous work because it considers the integration of multiple layers in the hardware and software systems, it supports the Data Distribution Service and CORBA Component Model of the Object Management Group, and it combines real-time, fault tolerance and multi-level security. This effort addresses problems that are currently important for U.S. Navy programs such as FORCEnet and Open Architecture, and also are important in the commercial sector. Validation of the specification approach has been achieved by employing it within Raytheon’s application integration framework (AIF) testbed.

1. Introduction

The Navy has embarked on the design and construction of the next generation destroyer [1]. This twenty-first century ship will leverage the best available technology today and in the future. The program will provide, for the first time in surface combatant command and control, a fully integrated command and control capability for assets that include ship systems, sensors, communications, and weapons. The scale of this integration will push middleware technologies beyond current commercial capabilities. The ship’s Design Agent has identified the following risks inherent in current commercially available middleware as applied to Naval vessel computation infrastructures: (1) survivability of mission critical functions, (2) dynamic resource management driven by Quality of Service (QoS) requirements, (3) real-time performance of dynamic concurrent threads of execution, and (4) Multi-Level Security (MLS) requirements for secure dynamic resource management, secure application interaction, and secure streaming data.

The current state-of-the-art for the Naval Total Ship Computing Environment (TSCE) resource management architecture uses fixed static allocation of resources in support of predefined mission capabilities. These mission capabilities are based on a design-time estimate of how operational conditions may change along expected boundaries. At run-time, once a resource is allocated, it remains allocated for the duration of the mission, and can’t be freed and reallocated to support another mission capability. When all resources have been allocated, no further mission capability is available, because all resources are effectively locked. This limits a commander’s ability to adapt to conditions that vary from the original system design. It also requires significant design-time analysis be performed to develop the allocation strategy, which must be repeated even for relatively small system upgrades. Finally, design-time analysis becomes increasingly less accurate and more expensive as the complexity of the system increases.

Failures in the onboard network also have mission-limiting effects, due to partitioning and static allocation of resources. There may be adequate resources in other locations onboard, capable of performing the required tasks of the failed elements. Static allocation strategies prevent these replacement resources from being used, despite their availability. Thus, it is desirable for resource allocation to be dynamically performed and modified in response to faults, to changes in mission requirements, or to workload distributions not matching the original mission-planning model.

Reallocation of resources in a dynamic TSCE is complicated by MLS policies imposed on allocation and use of resources. In advanced scenarios, in order for dynamic reallocation to be successful, the ability to dynamically select among alternatives across ship compartments and to migrate resources across security domains is needed. This capability overcomes limitations placed on resource management in a fixed security environment, where resources are restricted to a given security or physical compartment.

1 This work has been supported in part by funding from the DARPA ARMS Program. Approved for Public Release, Distribution Unlimited
It allows for resources to be made available to a mission capability based on the security domain membership and rules specified in the security policy.

The DARPA Adaptive and Reflective Middleware Systems (ARMS) program [2] is concerned with problems deemed too uncertain and difficult to address during typical Engineering Design Modeling (EDM) phases of Naval development programs. The ARMS program is pushing the state-of-the-art of several technologies focused on addressing these difficult challenges. The technology foci for the ARMS program include fault tolerant real-time middleware, MLS as a constraint, and shipboard-scale dynamic resource management.

This paper focuses on the resource management approach being developed in the ARMS program. Previous work in resource management can be found in numerous projects, including DeSiDeRaTa [4], DQM [5], Globus [6], HiPer-D [7], Q-RAM [8], and QuO [9]. These research projects have concentrated on (1) specifying characteristics of resources, resource usage policies and real-time software systems, (2) monitoring resource status and utilization, (3) reasoning about effective ways to allocate resources, and (4) controlling how resources are allocated to users and processes. The ARMS approach to resource management differs from, and extends, previous work because it considers the integration of multiple layers in the hardware and software systems, it supports the Data Distribution Service and CORBA Component Model of the Object Management Group (OMG) [3], and it combines elements of real-time, fault tolerance and multi-level security.

To first approximation, the DARPA ARMS resource management middleware (RMM) operates as follows. The features of the resources to be managed are characterized. Additionally, the properties, resource needs and performance requirements of the software systems are characterized. The characterizations of resources and software systems may be performed statically, as in [4], or dynamically, as in [5]. Given the characterizations, the RMM finds and enacts an initial (feasible) allocation of resources to software systems. Once the application systems are operational, the RMM monitors resource status and utilization, and software system resource needs. When the monitoring data indicate that the performance of the system needs to be improved, a reallocation is planned and enacted.

This paper will focus on a specification approach for the characterization of the software systems and hardware resources to be managed. While several approaches to this problem have emerged [see [10] for an overview], no standard specification approach has been defined. This is currently an important problem for transformational U.S. Navy programs such as FORCEnet and Open Architecture, and is also an important problem in the commercial sector. The Object Management Group (OMG) has a number of related initiatives in various stages of development, including the UML Profile for QoS, QoS for CORBA Component Model (CCM), and Data Distribution Services (DDS) for real-time. Additionally, the Application Management Services (AMS) RFP is currently being developed within the Command, Control, Communications, Computers, Intelligence (C4I) Domain Task Force of the OMG; while the AMS effort intends eventually to encompass the specification of information needed for dynamic resource management, its current scope does not encompass all of the issues addressed in this paper.

The remainder of this paper presents our approach for specification of the system properties needed for multi-layer resource management (MLRM). The paper is organized as follows. Section 2 describes the MLRM architecture and middleware that have been produced in the DARPA ARMS program. Section 3 covers the specification language that is used for MLRM. A presentation of how the specification language has been validated within Raytheon’s official testbed of the DARPA ARMS program is provided in Section 4.


This paper presents a specification approach that has emerged from the DARPA Adaptive, Reflective Middleware Systems (ARMS) program, which is producing multi-layer resource management middleware (MLRM). As shown in Figure 1, the MLRM approach supports a multi-layer system and resource hierarchy, which involves three layers: (1) the mission and system wide layer for coarse level global allocation, (2) the collections layer (resource pools and application strings) for fine-grained allocation within/using a subset of resources, and (3) the individual units layer (nodes, components, and applications) for allocation at an individual resource level. At the mission layer, the focus of concern is on managing the

---

2 Abstractly, an application string represents a software aggregation supporting one operational capability; essentially, it captures the notion of an end-to-end task within a mission.
operational capabilities that are required to support the current mission over the entire available infrastructure. Multiple distinct operational capabilities typically are required for each mission. At the collections layer, aggregations of interoperable hardware and software resources are managed as an integrated group to meet specific operational capabilities. At the individual units layer, resources and applications/components are managed to achieve various performance and quality of service (QoS) levels. The hierarchical software system is overlaid with multidimensional quality-of-service (QoS) requirements, which include the dimensions of real-time, fault tolerance, and security. In addition to presenting the specification approach, the authors will discuss tools that support the approach, and will discuss common problems that are important to consider as one develops a specification approach.

The primary components of the MLRM architecture are depicted in Figure 1. At the infrastructure layer, the infrastructure allocator (IA) is responsible for allocating individual application strings to particular resource pools. A resource status service (RSS) provides status and utilization information about resources and software systems to the IA, as well as to all other components that need such information. Each resource pool has an associated pool manager (PM), that oversees the allocation of all applications and application strings assigned to the pool by the IA. Resource allocation decisions for elements of application strings (i.e., applications) are made by a resource allocator (RA), which consults an allocation analyzer for feasibility analysis and optimality analysis. At the resource layer, resource provisioners allow the PMs to enact resource allocation decisions on hosts and networks. Additionally, the security infrastructure provides information about security policies and also enforces these policies. Once the mapping of applications to resources is enacted, localized monitoring and management of individual and end-to-end performance metrics are made by application managers.

Figure 1. Architecture of the multi-layer resource management middleware.
3. The Specification Language

The resource management system described in Section 2 requires descriptions of the system to be managed. This section describes the specification approach that the DARPA ARMS program has produced for this purpose. Section 3.1 presents our approach for the specification of software systems, and our approach for the specification of the properties of resources is described in Section 3.2.

3.1 Software Systems

To support MLRM, our specification approach allows several abstraction layers of software systems to be described. Section 3.1.1 describes the approach for specification of properties of application strings. The specification of properties at the individual units layer is discussed in Sections 3.1.2 and 3.1.3. Section 3.1.2 discusses the specification of properties of executable applications, and the specification of the properties of tasks (threads within an executable application) is discussed in Section 3.1.3.

3.1.1 Application Strings

An application string (AS) is a collection of QoS-constrained applications. The schema that depicts the attributes of an AS specification are shown in Figure 2. An AS specification includes the name of the string and a textual description of the string. These are followed by a reference to the AS type. An AS type describes the order in which the constituent applications of an AS must be started. To support computing systems that may operate in more than one mode, performance modes may be described. Each performance mode is characterized by its performance class, execution time, utility, importance, software reliability, and may have an optional critical path.

A performance class is a discrete value that indicates the level of assurance that application performance requirements will be satisfied. GUARANTEED means MLRM has exclusively reserved resources required to satisfy the specified level of performance and has performed some analysis to predict that performance will be satisfied in all cases/dimensions that the analysis technique addresses. PLANNED means that MLRM has allocated shared resources that on average will satisfy the specified level of performance, however, there could be operational periods where performance is less than desired. BEST EFFORT means that MLRM does nothing to ensure performance and, in fact, actively works to make sure the application does nothing to harm applications in the GUARANTEED and PLANNED classes. Utility characterizes the benefit or value that will accrue to the system when an application string operates in a particular performance mode. Importance (or criticality) indicates the relative level of mission importance or value. Its possible values are taken from the following set, whose elements are listed in decreasing order of importance: {SAFETY CRITICAL, MISSION CRITICAL, MISSION SUPPORT, QUALITY OF LIFE}. Software reliability characterizes discrete levels of required availability for applications. HIGH_AVAILABILITY means proper functioning after N faults, where N is configurable. MINIMAL OUTAGE means that an application string may be unavailable for some time T, where T is configurable. UNBOUNDED_OUTAGE means the system will not do anything to meet a specific reliability metric. The optional critical path definition associates a deadline with a segment of the application string.

Following the description of the performance modes of an application string, applications (executable programs) that constitute the AS are described. As shown in Figure 2, a reference to an application descriptor is provided, followed by a required security domain for the application (thus allowing an application string to span multiple security domains), the initial location where the application should execute, and, optionally, the number of replicas of the application that should be started by the MLRM. Additional attributes of applications are explained in the following subsection.
3.1.2 Applications

The attributes of applications, which are grouped to form application strings, are presented in this section. As shown in Figure 3, softwareInstances (i.e., executable applications) are logically grouped into softwareConfigurations to facilitate file management. Each softwareInstance is characterized by its name, a reference to its software type, an importance, a collection of tasks (independent threads), and a description of properties relevant to execution.

The software type, represented in the schema by swTypeNamed Reference, contains the items shown in Figure 4. Configuration information includes vendor, version number, and a description of the service provided. The processSpecification information describes the hostConfigurations on which the application can execute. Finally, the tasks that execute within the application are described. The features of the tasks are described in the following subsection.

Figure 2. The schema for application string descriptions.
Figure 3. Schema for a collection of application executables.
Figure 4. The schema for attributes of an executable application.
3.1.3 Tasks

As explained previously, each executable application may contain multiple threads of execution, which are called *tasks*. The attributes of tasks that are modeled in the specification files are described in this section. As shown in Figure 5, the five types of tasks that can be described are *periodic, event reaction, event generation, control*, and *human* tasks. Each task is characterized by its *minimum, nominal (expected) and maximum duration* (execution time), *behavior (period or minimum interarrival rate)*, and *performance modes*. Figure 6 shows the structure of the `softwarePerformanceMode` for tasks, which consists of the traditional real-time aspects, plus an optional `pubOrSubSpec`, to support OMG’s DDS publish-subscribe communication paradigm, which is used heavily within the DARPA ARMS and TSCE contexts.

If a task is a *publisher*, the following attributes are described. An optional *topicNamedReference* describes the size of the topic to which publications are made. Similarly, *Count* describes the number of topic messages published in each publication. The attribute *publishStyle* can have one of the following values: {multicast, broadcast, unicast, RMI_1way (1/2 duplex), RMI_2way (full duplex), mailbox}. The *commSpecNamedReference* field refers to the security specification, which describes access privileges (read, write, or read-write), indicates whether communications may span domains, provides a trusted interface name, and names the physical domain in which the communication may originate. Similarly, for *subscriber* tasks, the specification contains a *topicNamedReference, QoSParameters*, and a *commSpecNamedReference*.

![Figure 5. The schema for types of tasks and their attributes.](image-url)
Figure 6. The schema for software performance modes.
For a publisher, the **QoSParameters** fields of a *pubOrSubSpec* specification are durability, persistence, reliability, strength, and timeout. The **durability** property is the length of time that past issues are kept for subscribers that join the network after publication has begun. This parameter specifies how long, in milliseconds, the publications will be kept. Because these published issues are maintained in memory, it has resource implications and, therefore, the developer should use it with care. The **persistence** property is the length of time for which the issue remains valid after it is published. After this time has elapsed, the issue is considered invalid and will not be delivered to subscribers. The parameter is in milliseconds. Note that this property is relevant only when there is more than one application publishing a topic. The **reliability** property specifies the type of reliability this publisher should adhere to, either ALL or BEST_EFFORT. ALL is a guaranteed delivery by the publisher, who is notified that the issue was delivered; BEST_EFFORT is a best effort delivery, in which the publisher is not notified about whether the issue was delivered. This property must be set. The **strength** property is the priority assigned to a published issue relative to other issues of the same topic. This property allows for arbitration among publishers as follows: if the same topic is published by multiple sources, the one with the highest strength is received by the subscribers. The **timeout** property, specified in milliseconds, specifies the maximum elapsed time the publisher will wait to publish an issue. If its reliability property is set to guaranteed, the publisher maintains a queue of published issues. Issues are removed from the queue only when a positive acknowledgement is received from every subscriber; if a negative acknowledgement is received, the publisher republishes the issue. If the queue is full, the publisher will wait up to the time specified in the timeout parameter to put the new issue on the queue. If the time elapses, a publish exception will be thrown.

For a subscriber, the **QoSParameters** fields of a *pubOrSubSpec* specification are reliability, deadline and minimum separation. The **reliability** property specifies the type of reliability this subscriber should adhere to. There are two types of reliability: ALL and BEST_EFFORT. BEST_EFFORT reliability does not incur retries by the publisher, and involves the least overhead. It is used when the subscriber can afford to miss an issue (e.g., when publications are frequent and periodic). The subscriber is notified via an exception if an issue is not received within the specified deadline. The reliability property must be set. The **deadline** property is the time, in milliseconds, within which the subscriber expects to receive a new issue. If the deadline expires without an issue being published, the messaging service will notify the subscriber. The **minimum separation** property is the minimum allowable elapsed time, in milliseconds, between arrivals of issues. This property allows a subscriber to protect itself from high-rate publishers. Note that this property is only relevant for subscriptions whose reliability property is set to BEST_EFFORT; otherwise it is ignored.

### 3.2 Hardware Systems

An MLRM system not only requires specification files that describe the software systems, but also requires that the hardware configuration be specified. Thus, the specification language also allows the specification of resource pools, of hosts, and of networks. The distributed hardware system is viewed as a collection of resource pools. Each *resource pool* is a collection of host resources and network resources. As shown in Figure 7, the specification for each *host* resource includes number of CPUs, memory capacity, display surfaces, security domain and operating system version.

The **network** specification (see Figure 8) consists of descriptions of switches in the network, their interconnections (interfaces) and subnets serving hosts by edge routers. A network topology consists of a **switchlist**, to denote a sequence of switches (an attribute in a *switch* indicates whether the switch is a layer-2 or layer-3 switch), **L3InterfaceList**, to denote a sequence of layer-3 interfaces, **L2InterfaceList**, to denote a sequence of layer-2 interfaces, and **subnetToEdgeInterfaceList**, to denote a sequence of subnets. Each subnet is served by an edge router and consists of at least one host. A *switch* is characterized by such attributes as whether it is an edge router, its *internalID*, *managedElementID* (its management interface), information required to login to the switch and to configure it, physicalLocation, switchID (e.g., serial number for inventory purposes), and the operating system of the switch. The attributes of an **L3Interface** include: its *InternalID*, *routerInternalID* (the layer-3 switch it belongs to), *interfaceID* (usually in the form of technology (e.g., Ethernet), slot #, port #), its *IPAddress*, *IPSubnetMask* to determine what subnet the IP address belongs to, and the interface (link) capacity in terms of in and outbound bandwidth (bits/second). The attributes of an **L2Interface** include: its *InternalID*, *switchInternalID* (the layer-2 switch it belongs to), port (its physical port), **L2Address** (Layer-2 (MAC) address), *vlanIDs* (the virtual LANs it belongs to), its
capacity in terms of in and outbound bandwidth (bits/second) and a flag isRemoteL2 (to determine whether the remote side of the interface is an L2 or L3 interface, as we support a hybrid layer-3 and layer-2 topology). The attributes of subnetToEdgeInterface are: its InternalID, the IP address of the subnet, IPSubnetMask, in and outbound bandwidth (bits/second), edgeRouterInterfaceID (the internal ID of the switch that serves the subnet consisting of hosts), and the IP address of the switch. We also employ a variation to this subnetToEdgeInterface modeling to capture the in/out bandwidth on the links to hosts, if the hosts are directly connected to a switch.

Figure 7. The schema for a host hardware profile.
Validation within the DARPA ARMS Testbed

Validation of our specification approach has been achieved by employing it within Raytheon’s application integration framework (AIF) testbed for DARPA ARMS challenge problems. This section provides an overview of the challenge problems and of the testbed, and describes how the specification language has been used successfully in this context.

Experiments and challenge problems have been defined to measure the success of Phase I of the DARPA ARMS program and to assess the likelihood that the resulting technology will help to further the capabilities of the Total Ship Computing Environment (TSCE). The experiments are as follows: (1) enhance the TSCE so it automatically and accurately adapts to dynamic changes in mission requirements and environmental conditions; (2) leverage dynamic resource management to increase TSCE performance to increase ship survivability; and (3) increase TSCE capabilities in the case of failures arising from battle...
damage or computing system failures. Additionally, the following challenge problems have been put forth: (1) achieve continuous operation by adapting to changes in environment, mission, and resource availability; (2) perform dependency driven reconfiguration by using configuration optimization strategies to increase overall capacity; and (3) provide adaptation to failure to handle errors and failure conditions not part of challenge problem (1). The remainder of this section discusses the software and hardware testbed that is being used to validate that the ARMS MLRM middleware passes the gate experiments and solves the challenge problems. The sufficiency of the specification language is shown by using it to model this testbed.

Validation of the MLRM middleware, including the specification language, is performed in the context of the application integration framework (AIF) testbed, which is depicted in Figure 9. Experiments are performed as follows. An Experiment Controller (EC) executes test scenario scripts that cause workload generators (WLGs) to be executed under control of the MLRM middleware. Experiments are performed by varying workload through one or more combinations of the following: (1) adding mission capability (i.e. application string(s)), which deploys and executes workload generators, (2) removing application string(s), and (3) injecting faults into workload generators. Table 1 lists the specific workload generator types that are used in the experiments.

Figure 9. The AIF context.
Table 1. Workload generator types in the AIF testbed.

<table>
<thead>
<tr>
<th>Workload Generator Type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment Controller</td>
<td>Controls and monitors experiment execution by managing Operational Plans, Mission Capabilities and Policy Modification.</td>
</tr>
<tr>
<td>Critical Information Transfer within Infrastructure</td>
<td>Models intra-infrastructure data transfer. These are considered critical components. As such, they are characterized as requiring very high priority and minimum response time. Once started, their execution will persist, run continuously, and will not be interruptible. 4 variations: CX-1..CX-4</td>
</tr>
<tr>
<td>Effectors Management</td>
<td>Models processing by manipulating extra-system environments, such as weapons. These components will remain relatively static with respect to their instantiation. Nominally, they will either be brought online, left online, or remain inactive. They are considered critical, having high priority and rapid response. 13 variations: EFF-1..EFF-13</td>
</tr>
<tr>
<td>Emergency Response</td>
<td>Models a component that might handle abnormal or emergency situations. These are expected to have low periodicity and infrequent occurrence. When active, however, they are expected to have high priority. There will likely be many of these, but they won’t do much unless an emergency is injected into the experiment. 1 variation: ER</td>
</tr>
<tr>
<td>Environment Detection</td>
<td>Models processing of extra-system inputs, such as sensors. 15 variations: ED-1 .. ED-15</td>
</tr>
<tr>
<td>Externally Controlled Information Transfer</td>
<td>Models extra-infrastructure controlled data transfer, such as off-board communications. These are controlled by an external component. They likely will be many of these running at varying priority and periodicity as the experiment dictates. These will require hard real-time response times. Periodicity will be minimal to moderate, as required by the given experiment. The performance cycles for these variations will all be below 5 seconds. 10 variations: XCX-1 .. XCX-10</td>
</tr>
<tr>
<td>Planning</td>
<td>Models plan generation and manipulation processing. 5 variations: PLAN-1 .. PLAN-5</td>
</tr>
<tr>
<td>System Management and Monitoring</td>
<td>Models extra-infrastructure executive processing. The executive nature of these components will require hard real-time response times. Periodicity will be minimal to moderate, as required by the given experiment. The performance cycles for these variations will all be above 5 seconds. 4 variations: SMM-1 .. SMM-4</td>
</tr>
</tbody>
</table>

To conduct experiments, the WLGs were combined to form sets of application strings that increase in complexity, in order to test the capabilities of the MLRM. The following test cases have been created:

1. **single pool test** - all WLGs run in a single resource pool;
2. **single pool test with a critical path** - same as test (1), with the addition of a critical path defined;
3. **two pool test** - extends test (1) by adding WLGs and spreading over 2 pools that are in the same security domain;
4. **two pool test with a critical path** – test (3) with the addition of the same critical path as 2;
5. **two pool test in 2 security domains** - same WLGs as in test (3), but the second pool is in a different security domain;
6. **two pool test in 2 security domains with a critical path** - same as test (5), with same critical path as defined in (2);
7. **three pool test in 2 security domains** - extends test (5) to add WLGs in a third resource pool; and
8. **three pool test in 2 security domains with a critical path** - same as test (5), with same critical path as defined in test (2).
Figure 10. Use of the specification files in a MLRM context.

Specification files have been produced for each of the above test cases. As shown in Figure 10, these files are used by the MLRM to make resource allocation decisions. Figure 11 summarizes the specification files that were created for the “single pool test” that was described above. As shown in the figure, 50 specification files describe the test. Figure 12 summarizes the specification files that have been created to describe the entire suite of tests. Note that these tests involved 500 instances of 50 different WLG types and resulted in more than 1000 specification files.

The specification files were created by following the process shown in Figure 13. This process involves the creation of XML schema (described in Section 3 this paper) by using the XMLSpy tool, the creation of XML specification files that describe the actual tasks (e.g., WLGs) and resources to be managed by MLRM, and the parsing and validation of the XML files, which relies on the XMLSpy and Xerces tools.
Testbed files:
- AIF_resourcePool.xml
- AIF_hardwareConfiguration.xml
- AIF_networkTopology.xml
- securityCommSpec.xml

App String files:
- AIF_1pool.1.appstring.type.xml
- AIF_1pool.1.appstring.instance.xml

* WLG files:
ed-2 WLG files:
- *-1.wlg.type.xml
- *-1.1.wlg.instance.xml
- *-1.topics.xml
- *-1.1.wlg_char.xml
ed-1 WLG files:
- ed-1.wlg.type.xml
- ed-1.topics.xml
- ed-1.1.wlg.instance.xml
- ed-1.1.wlg_char.xml

Same 4 files for all scenarios

Unique type and instance for each (sub)scenario

Total: 4 test bed + 2 app string + (4 files * 11 wlg types)
= 50 XML files

Figure 11. Validation via a single pool test.

<table>
<thead>
<tr>
<th>WLG Type</th>
<th># types</th>
<th># instances per type</th>
<th># instances</th>
<th># swConfig files</th>
<th># swType files</th>
<th># topic Manifest files</th>
<th># WLG char files</th>
<th>total files</th>
</tr>
</thead>
<tbody>
<tr>
<td>env detector (ed)</td>
<td>15</td>
<td>10</td>
<td>150</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>150</td>
<td>330</td>
</tr>
<tr>
<td>ship monitor and mgmt (smm)</td>
<td>5</td>
<td>10</td>
<td>50</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>110</td>
</tr>
<tr>
<td>planning and decision (plan)</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>configuration optimization</td>
<td>5</td>
<td>10</td>
<td>50</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>50</td>
<td>110</td>
</tr>
<tr>
<td>externally controlled transfer (xct)</td>
<td>10</td>
<td>10</td>
<td>100</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>effector management (eff)</td>
<td>13</td>
<td>10</td>
<td>130</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>130</td>
<td>286</td>
</tr>
<tr>
<td>total files</td>
<td>50</td>
<td>500</td>
<td>500</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>500</td>
<td>1100</td>
</tr>
</tbody>
</table>

Other XML files:
- app string type files
- app string instance files
- ware configuration files
- resource pool files
- security comm spec files

total XML files 1141

Figure 12. XML files produced to describe the AIF testbed.
6. Conclusions

This paper presents a solution to the problem of specification of the properties of software systems and hardware configurations that are relevant to multi-layer resource management. This important problem is being addressed in several contexts today, including the standardization efforts of the Object Management Group and programs of the US Navy. The solution presented here has been evaluated within Raytheon’s Application Integration Framework, a testbed that is representative of complex real-time systems. The results of this effort are already having an impact on future defense systems and on commercial standards. Ongoing work for this project includes algorithmics and middleware research, transition to Navy ship platforms, and development of standards for specification and for other aspects of resource management.

7. References


