Secure Middleware for Defence Applications

Dr. Ramesh Bharadwaj
Center for High Assurance Computer Systems
Naval Research Laboratory
Washington DC 20375
USA

Ramesh@itd.nrl.navy.mil

Dr. Marc Born
Fraunhofer FOKUS
Berlin
GERMANY

born@fokus.fraunhofer.de

Mr. Rudolf Schreiner
ObjectSecurity, Ltd.
Cambridge
UK

Rudolf@objectsecurity.com

ABSTRACT

Achieving robust and secure system interoperability over Mobile Wireless Networks poses a number of daunting challenges: (1) Ensuring robustness and survivability in the presence of network jamming, transient faults, frequent node failures (e.g., due to the batteries on a PDA running out,) and rapidly changing network topology and connectivity. (2) Achieving acceptable performance and providing the necessary Quality of Service (QoS) guarantees over low-bandwidth, unreliable links. (3) Ensuring system integrity in the presence of malevolent code such as worms and viruses, Trojan horses, intruders, eavesdroppers, and malicious attacks. The goal of the Secure Infrastructure for Networked Systems (SINS) middleware project is to provide secure, efficient, and robust distributed system interoperability, to reduce total ownership costs, to allow quick and easy system upgrade and reconfiguration, to lower the impact of Commercial Off The Shelf (COTS) upgrades, and to reduce compatibility problems. Target applications for SINS include information network situational awareness, networked C^2 for combat applications, and Unmanned Aerial Vehicle (UAV) swarms. The Secure Middleware Platform, based on Model Driven Architecture (MDA) and the CORBA Component Model (CCM), has proved to be very effective for the development, deployment, and maintenance of distributed applications. Model-based application development greatly reduces effort by providing application developers abstractions of the underlying middleware and protocols.

1.0 INTRODUCTION

Defense C^4ISR systems are increasingly being required to interoperate over platforms and networks with widely varying computational power and data bandwidths. These applications, platforms, and networks will form the architectural framework – termed as “Network-Centric Systems” – for the information age, integrating sensors, platforms, weapons, and command and control with the war fighters into a networked, distributed combat force.

The military environment is very demanding, because interactions are highly dynamic, networks are disadvantaged (e.g. low bandwidth), heterogeneous, and often require ad-hoc interoperability with coalition partners. Also, there are various discretionary and mandatory access control restrictions on

Bharadwaj, R.; Born, M.; Schreiner, R. (2006) Secure Middleware for Defence Applications. In *Military Communications* (pp. 18-1 – 18-6). Meeting Proceedings RTO-MP-IST-054, Paper 18. Neuilly-sur-Seine, France: RTO. Available from: http://www.rto.nato.int/abstracts.asp.

exchanged data, with different needs-to-know predicated on data currency. Therefore, requirements for such a large, complex distributed system are manifold, and include interoperability, flexibility, modularity, rapid and cost-effective development, deployment, and migration even in multi-company and multi-national contexts, easy (re)configuration and efficient maintenance, and information assurance. Moreover, the whole system life cycle needs to be considered, as well as adaptability and quality of service (QoS) at different levels. From a security perspective, adequate protection of communications (including wireless communication), appropriate authorization and audits, and protection from attacks, e.g. denial of service, are mission-critical.

Today, the main approach for the development of distributed applications is the use of middleware to abstract from the underlying network technology, but even the best available middleware platforms and modeling standards do not meet all the requirements of defense applications.

2.0 SECURE MIDDLEWARE

It is widely acknowledged that we need new ways to tackle complexity, the Achilles heel of system vulnerabilities. These demanding requirements, the need to cope with increased uncertainty and dynamic changes in an application's environment, and the requirement of greater flexibility, have created the need for a paradigm shift in terms of how applications for Mobile Wireless Networks are designed, implemented, deployed, and maintained. In this section, we explore two approaches to the construction and deployment of secure middleware. One approach is SINS being developed at the Naval Research Laboratory; another is secure middleware that was developed in conjunction with the EU-COACH project.

2.1 SINS

Secure Infrastructure for Networked Systems (SINS) is distributed middleware being developed at the US Naval Research Laboratory to assist developers build applications without the need to hand-craft code that addresses the above concerns. SINS provides developers the ability to specify situation-awareness, dynamic interoperability, reconfiguration, and QoS requirements, in a declaratively manner. The middleware will attempt to satisfy these requirements by a judicious choice of underlying components and weaving-in the necessary code. We ensure System Integrity in SINS by authentication and authorization mechanisms, providing confidentiality and integrity of transmitted information, design of security protocols for fast and easy system reconfiguration, and the harnessing of a model checker for safety and security policy enforcement. We address performance by providing dynamically determined agent routing patterns, a mechanism for flexible event handling and propagation, and highly efficient transmission of relevant information (e.g., by only transmitting changes to an object rather than the object itself.) Finally, we achieve robustness by providing the Secure Operations Language (SOL) for specifying interaction and coordination, an Agent Creation Framework that checks application code for completeness and consistency, mechanical proofs of safety properties and compliance with local security policies, and determination of the emergent behavior of an application that is comprised of many interacting components. Finally, we introduce the novel concept of security agents that act as mini-firewalls between applications and the Operating System (OS) resources. The goal of the SINS middleware project is to provide secure, efficient, and robust distributed system interoperability, to reduce total ownership costs, to allow quick and easy system upgrade and reconfiguration, to lower the impact of COTS (Commercial Off The Shelf) upgrades, and to reduce compatibility problems. Target applications for SINS include information network situational awareness, networked C2 for combat applications, and Unmanned Aerial Vehicle (UAV) swarms.

In order to meet current systems engineering challenges such as pervasive and ubiquitous computing, one has to adopt model-based approaches to the development of distributed applications. One answer to the systems integration problem is the use of the synchronous paradigm for component integration and coordination, where developers are provided with an abstraction that respects the synchrony hypothesis, i.e., one may assume that an external event is processed completely by the system before the arrival of the next event. Based on the synchronous model, the Secure Operations Language (SOL) [2] is designed for the integration of high assurance systems. The utility of SOL hinges upon the fact that it is a verifiable language. Programs in SOL are amenable to fully automated static analysis techniques to ensure compliance of a system with application specific requirements and local or global security policies. For a detailed treatment of the language SOL and its formal semantics, the reader is referred to [2, 3, 4].

SOL agents execute on a distributed run-time infrastructure called SINS [3]. A typical SINS implementation comprises one or more SINS Virtual Machines (SVMs), each of which is responsible for the instantiation of SOL agents on a given host. An application in SINS comprises a set of software agents that avail of services provided by SINS virtual machines running on disparate hosts over a network. SINS provides mechanisms for the creation, deployment, and migration of agents, in addition to protocols for inter-agent communication and synchronization. One might wonder how a synchronous language such as SOL can be implemented on widely distributed systems where there is inherent asynchrony. The answer is that SINS uses the Spread toolkit [1] which implements the necessary protocols to provide a high performance virtual synchrony messaging service that is resilient to network faults.

2.2 CCM/CORBA

We propose to extend current standards in middleware and modeling, including the Object Management Group (OMG) standards CORBA Component Model (CCM) and Model Driven Architecture (MDA), into a Secure Middleware Platform that will serve as the information grid for Network-Centric Warfare. Our architecture is based on several simple principles: Modelling of functional and non functional aspects of the application, separation of concern, separation of business logic and infrastructure, abstraction from underlying protocols and component based system development. In our architecture, an application comprises three clearly separate sub-systems: Containers provide a distributed runtime environment and infrastructure; Service Components implement non-functional requirements and Application Components implement the functional properties of the application, i.e., the business logic.

The container is an enhanced version of the CCM container. It provides a complete runtime environment for CORBA components. The container provides application interfaces for the component life cycle, for communication between components and for the setup of connections between the components. It also provides several basic services, such as naming, and a standardized configuration of a single component and the application as a whole. The OMG CCM container supports synchronous and asynchronous communications, the invocation of operations on components and the exchange of messages or events between components. We have added streams as a third communication paradigm. Streams are an abstraction of the underlying streaming protocol, used for example in high performance data exchange or for voice/video communication. CCM is conventionally implemented on top of a CORBA Object Request Broker (ORB). Our current implementation of CCM is on an ORB with enhanced support for security protocols and additional security infrastructure. The network protocol we currently use, TCP/IP, is not ideally suited for all communication media, e.g., for wireless communication; however, the architecture supports transparent replacement of the protocol stack by other protocols or even replace the CORBA ORB by alternate middleware protocols. This middleware neutrality allows application to migrate unmodified to different environments or protocols, e.g., to cross layer protocols, secure multicast protocols like Spread, or the Data Distribution Service (DDS).

The components implement the business logic, i.e., the functional parts of the application. They could be developed using conventional software engineering techniques, but we recommend the Object Management Group OMG's Model Driven Architecture for the development process: Instead of implementing each component as source code, application developers instead define UML model of the components, their interfaces and their interactions, and automatically generate a large part of the

UNCLASSIFIED/UNLIMITED

component source code and meta-data directly from the model, using model transformation. Only portions of the business logic need to be written manually. We also use another component of the MDA, the Meta Object Facility (MOF), for data- and information- modelling, and for the automatic generation of data storage and interfaces. This approach, based on models and model transformation, greatly simplifies the application development process.

A very important, but often neglected aspect of the application is still missing: The non-functional requirements, for example, quality of service, and security. These play a very important role in military systems. It is common practice to embed such non-functional requirements directly into the source code, both in monolithic and component based software architectural approaches. However, this approach has several disadvantages; for example, it reduces component reusability and binds an application to a specific environment. In the Secure Middleware Platform, we are able to model and implement non-functional aspects very much like functional aspects, using MDA/UML and special-purpose service components. Service components intercept communication between components and directly access the underlying network infrastructure, e.g., to set network properties. For instance, we modelled and implemented a service for bandwidth reservation for high priority communication. The main advantage of service containers is that they seamlessly integrate security into the middleware platform using our OpenPMF Policy Management Framework. OpenPMF is a framework to define and enforce security policies in distributed systems. Also based on the MDA principle, an abstract model of the security policy to be enforced is created, and directly mapped to the concrete platform. For example, security policies for access control, or control of information flow in a multi-level secure environment, could be defined in the Policy Description Language of the Policy Management Framework (PMF) and loaded into a repository. During application start-up, the required policy is loaded into the service container and enforced by interceptors. For applications without online connections to the policy repository, it is also possible to directly generate code that enforces the policy.

Implementing non-functional aspects using service containers has proved to be very powerful. For example we have used our platform to implement a MILS[1]-CCM system, with containers running in different partitions at different security levels. OpenPMF was used to control the communication between components – within a single domain, and, using a CORBA domain boundary controller, across domains. This architecture provides for separation of concerns, separating the business logic and non- functional aspects.

A very important and often neglected function in distributed systems is the deployment of components on different hosts. This is often done manually by the administrator. The Secure Middleware Platforms include secure component deployment architectures, for distribution of application and service components to different hosts and containers, to load the policies defining the non functional properties, and to set up the initial connection topology.

3.0 CURRENT STATUS

Prototype implementations of the described platforms are available. Currently, the SINS platform is implemented in Java and uses the Spread toolkit for group communication and key management. Currently, three research and development groups are evaluating the middleware. As for the CORBA/CCM middleware platform, it currently consists of the CCM implementation Qedo (Quality of Service enabled distributed objects), MICO as the underlying ORB, with enhanced security support provided by OpenPMF. The platform was successfully evaluated on several demonstration projects.

[1]Multiple Independent Levels of Security

UNCLASSIFIED/UNLIMITED

UNCLASSIFIED/UNLIMITED

4.0 FUTURE WORK

Associated with SINS is the Secure Operations Language (SOL), a synchronous language supporting the behavioural specification of agents in addition to their composition and coordination. SOL has been extended with features (such as the notion of component failure) to support a more general architecture for the composition of reusable components for critical applications that must satisfy important security, real-time, and fault-tolerance requirements. We have documented an initial study in formal verification of architectural patterns in support of construction of dependable distributed applications. This initial study has shown that it is relatively straightforward to prove a safety property associated with a generic module that implements an architectural pattern. The efficacy of this approach is that more generic architectural patterns need only be verified once and then instantiated for each system in which they are utilized.

In the CORBA/CCM project, we have so far concentrated on low level aspects, such as interfaces to ORB-level security protocols, the infrastructure, interfaces between the container and the service components, and effective evaluation of security policies. The focus of our future work will be to exploit this infrastructure for different purposes, for example, to use the service containers for other QoS requirements, such as fault tolerance, and infrastructure for the management of large clusters. Another area for future work is safety, for example to provide support for defining and checking constraints. We currently view CCM as a static distributed computing environment, but components can also be managed and deployed as dynamic agents. We plan to use the CCM platform for implementing dynamic reconfiguration and self organization, especially in the context of ad-hoc networks.

5.0 CONCLUSION

The goal of the NRL dependable middleware project is to develop infrastructure to deploy and protect time- and mission-critical applications on a distributed computing platform, especially in a hostile computing environment, such as the Internet. Such an infrastructure may be used for developing secure confinement mechanisms for unreliable or untrusted COTS components.

The Secure Middleware Platform based on Model Driven Architecture and the CORBA Component Model has proved to be very effective for the development, deployment, and maintenance of distributed applications. Model-based application development greatly reduces effort, while improving code quality, by providing abstractions of the underlying protocols and middleware to the application developer. Also, middleware-neutral development eases migration of applications and seamlessly integrates non-functional requirements such as quality of service and security into the model-driven architecture.

6.0 REFERENCES

[1] Y. Amir and J. Stanton. “The SPREAD wide area group communication system.” Technical report, Johns Hopkins University, Baltimore, MD, 1998.

[2] R. Bharadwaj. SOL: A verifiable synchronous language for reactive systems. In Proc. Synchronous Languages, Applications and Programming, ETAPS 2002, Grenoble, France, April 2002.

[3] R. Bharadwaj. Verifiable middleware for secure agent interoperability. In Proc. Second Goddard IEEE Workshop on Formal Approaches to Agent-Based Systems, Greenbelt, MD, October 2002.

[4] R. Bharadwaj. A framework for the formal analysis of multi-agent systems. In Proc. Formal Approaches to Multi-Agent Systems, Warsaw, Poland, April 2003.

[5] S. S. Yau, S. Mukhopadhyay, and R. Bharadwaj. Specification, analysis, and implementation of architectural patterns for dependable software systems. In Proc. 10th IEEE Int'l Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2005), Sedona, AZ, Feb. 2005.

UNCLASSIFIED/UNLIMITED