

The Next Step - Applying the Model Driven Architecture to HLA

Shawn Parr

Russell Keith-Magee

Calytrix Technologies Pty Ltd

PO Box 1173, Bentley, WA 6982, Australia

+61 8 9362 5300

{shawn.parr, russell.magee}@calytrix.com

Keywords:

HLA, Interoperability, Reuse, Model Driven Architecture (MDA), Simulation Component Model (SCM), Open Standards, Unified Approach.

ABSTRACT: *The goal of the High-Level Architecture (HLA) is to promote interoperability and reuse between federates. In this context, HLA fills the role of middleware in the domain of distributed simulation. However, in many respects, HLA falls short of its promised potential. These problems can be attributed in part to the HLA standard itself, which leaves a number of key infrastructure issues unaddressed. This has led to interoperability issues between vendor RTIs. In addition, while HLA provides a number of key simulation services, the HLA community has not yet embraced emerging standards from other similar middleware environments (such as CORBA and J2EE).*

One currently emerging standard in middleware environments is the OMG's Model Driven Architecture (MDA). MDA is an important "next step" for component models, as it provides a mechanism to abstract the design and behavior of a component-based system away from the specifics of the platform implementation in which it will operate. This separation of design and implementation allows the developer to concentrate on the behaviour of the system, rather than the specific and often binding implementational issues associated with middleware services. This approach leads to enhanced interoperability and reuse characteristics in components developed using the MDA philosophy.

There is a strong correlation between the issues currently facing HLA developers and the problems solved in other middleware environments through the adoption of an MDA development approach. This paper will introduce the MDA architecture and the motivations behind its development. This paper will also describe a Simulation Component Model, and demonstrate how this SCM can be used to deliver an MDA approach and tool set for simulation development. Finally, this paper will demonstrate how an MDA approach to HLA development can address the challenges of evolving standards, interoperability and reuse that currently exist in HLA.

1. Introduction

While the High Level Architecture (HLA) has been a success in many respects, particularly in the areas of reuse and interoperability, there are a number of existing and new challenges facing the application and the future of the standard.

A number of improvements have been made to HLA since the initial release of the standard; however HLA has changed little in relation to other middleware domains. For example, UML based modeling, which is common in other software engineering domains, has not been widely adopted. Similarly, the movement to component models that has been observed in other middleware infrastructures (like J2EE and CORBA) has not been observed in the HLA community. Interoperability issues between the various standards (HLA 1.3 and IEEE 1516)

and RTI implementations also continue to be a source of frustration.

This situation has led a number of authors to examine the current state of HLA [1, 2, 3], with the aim of exploring how HLA can be extended to support new standards. While important work is currently underway to address some of the immediate interoperability issues [4], it is important that, as a community, we take a broader, more holistic view of how HLA can evolve, with the goal of maintaining an up to date HLA standard with application both within the defense sector and beyond.

In response to these concerns, a new initiative called the Model Driven Architecture (MDA) from the OMG [5] has been proposed as a possible direction for the HLA standard [1, 6]. This paper will introduce the motivation and concepts behind the Model Driven Architecture as well as a technical overview of the MDA process. This will be followed by an overview of how MDA applies to

HLA and how HLA can be realigned to MDA with the development of a component model and UML profile.

2. An Overview of MDA

The OMG's Model Driven Architecture (MDA) provides an approach for designing and building a component-based system that remains decoupled from the languages, platforms and middleware environments that are eventually used to implement the system. Central to the MDA philosophy is the concept that an organization should be able to model their systems once and then transition them over time as standards and infrastructure technologies evolve and change.

Figure 1 shows the OMG's vision of the Model Driven Architecture [5]. As this figure shows, the MDA has a very broad scope. As such, it is not a single standard or even a single concept, but rather an amalgam of various industry domains, open standards and modeling techniques (UML, MOF, XMI and CWM) that have been consolidated over the last few years to support a common direction and methodology. Nor would it be accurate to describe the MDA as a revolution, but rather an evolution that encompasses existing key technologies in order to address a number of pressing issues in the software industry, including interoperability, reuse, design and change.

At the centre of the MDA approach is the model. As the acronym implies, MDA development is driven from and maintained through a platform independent model. While design models are already used extensively in the initial design phases of projects (often utilizing UML notation), in a pre-MDA environment these are usually treated as a starting point to the final implementation. In contrast, the MDA advocates that the model *is* the system, and that all changes must go through the model.

In order to understand the emphasis on modeling, this section will examine the motivations behind adopting the

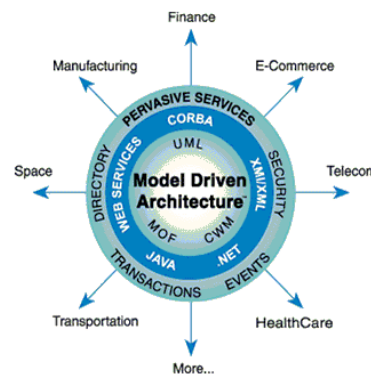


Figure 1: OMG Model Driven Architecture

MDA, followed by an overview of the design process and its underlying technologies. This will be followed by a discussion on how MDA can be applied to simulation.

2.1 Motivation for adopting the MDA

In order to appreciate the strong focus placed by the MDA on modeling and business process, it is important to understand the motivations behind the initiative.

Motivation 1: Managing change

One of the continual challenges facing the IT industry is the pace and frequency of change, whether that be evolutionary changes in standards or major infrastructure upgrades such as those imposed by new platform and infrastructure technologies like J2EE, CCM and Web Services. Figure 2 shows the adoption of platform technologies over the last decade. From this figure we can make three observations. Firstly, platform technologies tend to evolve and change. Experience would suggest that this occurs within a period of three to five years. It would be reasonable to assume that the technologies shown in this figure will continue to evolve. Secondly, as successful technologies become mainstream, the remaining technologies disappear. This presents a real challenge; while on the one hand there is a need to stay

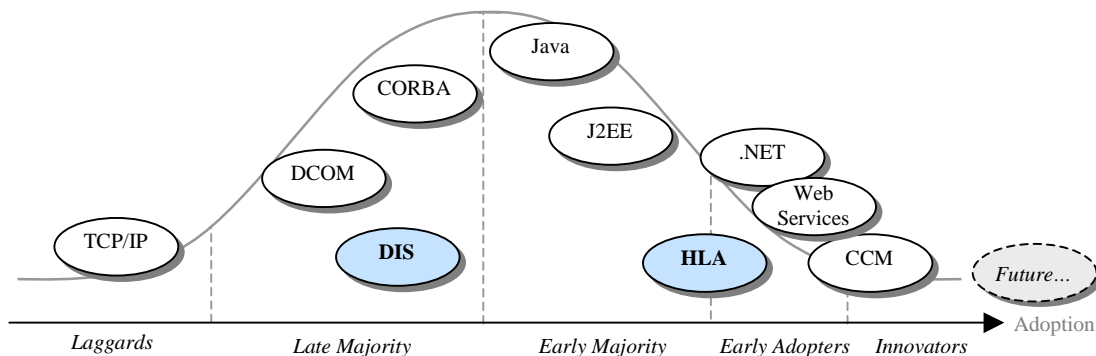


Figure 2: Technology adoption curve

current in order to maintain competitive advantage, there is a risk of adopting a dead-end or vendor specific technology that does not allow a system to move forward. Thirdly, as a technology becomes ubiquitous it will often form part of the core infrastructure used by the next generation of products, reaffirming the need to stay technologically current.

This constant change presents a real problem to an organization, which needs to both use and integrate their existing software while employing industry standards to provide *future proofing* for tomorrow's *next best thing*. Unfortunately, while developers appreciate that change will inevitably occur (and that this is a good thing), current development methodologies and tool sets do not adequately support this kind of change. In part this failing can be attributed to the way current design and development tools tend to be targeted at a single platform. If that platform changes or needs to be replaced then it is usually a manual exercise to repurpose the software. For example, in an enterprise scenario this might mean migrating a system from CORBA 2.3 to CORBA 3.0 (and the CORBA Component Model) [7] or from a Java platform using RMI to a J2EE container based model. A similar situation exists in the simulation arena, with groups having to support both the OMG's HLA 1.3 specification and the IEEE 1516 HLA standard, with new platform initiatives like the eXtensible Modeling and Simulation Framework (XMSF) [8] looming.

Motivation 2: Promoting the business model

Secondly, software systems tend to serve an organization best when they focus on the business needs first and the technology second [9]. However, as we have already seen, our current development environments tend to be very platform specific; so while a system design may start at a business level it will invariably end up as a programming exercise. At the end of the development process, only a loose coupling between the initial business constraints and the final platform-oriented design remains, making traceability between the now disconnected designs difficult. In practice this makes it difficult to see how a business change will impact the physical system. Conversely, because the business model has been altered to fit the target platform, it is difficult to transition or reuse the business aspects of the design with a new platform. Under these conditions it is the technical needs and not the business concerns that drive the development of systems.

Addressing these problems

The MDA directly addresses these problems by codifying and standardizing the steps required to take a model from design through to final implementation. Using the MDA approach, a system is defined by a *platform independent model* (PIM) that captures all the business and non-

business (security, performance) attributes of a system. The PIM and its related models becomes the core development artifact of the system. From the PIM, the MDA defines a process of refinement and transition that allows the model to be realized through a *platform specific model* (PSM) for a given platform. However, unlike current modelling environments that separate the design phase from the subsequent development and implementation tasks, the MDA process clearly defines how the PIM is realized via a PSM for any given platform without polluting or compromising the core design integrity with implementation issues.

This approach has two effects. Firstly, the design remains the central artifact of the system, ensuring that the business model captured in the PIM always reflects the actual system. Secondly, by not tying the design to any given platform, the PIM and its associated business logic can be transitioned over time as standards evolve and when the next big thing comes along.

The following section will describe in more detail the development process using the MDA.

2.2 Technical overview

The first step when developing an MDA model is to describe the system using a PIM. The PIM provides a complete description of the system's business and non-business attributes, without reference to any implementation or technical concerns. The PIM is captured using UML and associated metadata. Industry specific models are emerging in the form of UML Profiles, which provide a mechanism to capture additional business-oriented information in the model.

While the ultimate goal of the MDA is to allow a system to move directly from the PIM specification to the completed system, in reality this ambition is still some way off. As a result there are a number of intermediate steps required, through the definition of platform specific models (PSM), to refine the PIM for the targeted platform. As the design focus moves from the PIM to PSM there is a shift in emphasis from the business aspects at one end of the spectrum to the technical aspects at the other end. The PSM captures additional metadata, in the form of UML diagrams and *transformation rules*, that describe how the services requested at the PIM level can be realized in the targeted environment. A PIM may have multiple PSMs for multiple platforms.

The MDA also describes a number of *pervasive services*, such as security, transactions, and naming, that should be supported by all target platforms. These services are provided though the platform's container or component model. It is therefore the PSM's role to describe how a requested service at the PIM level is to be realized in the

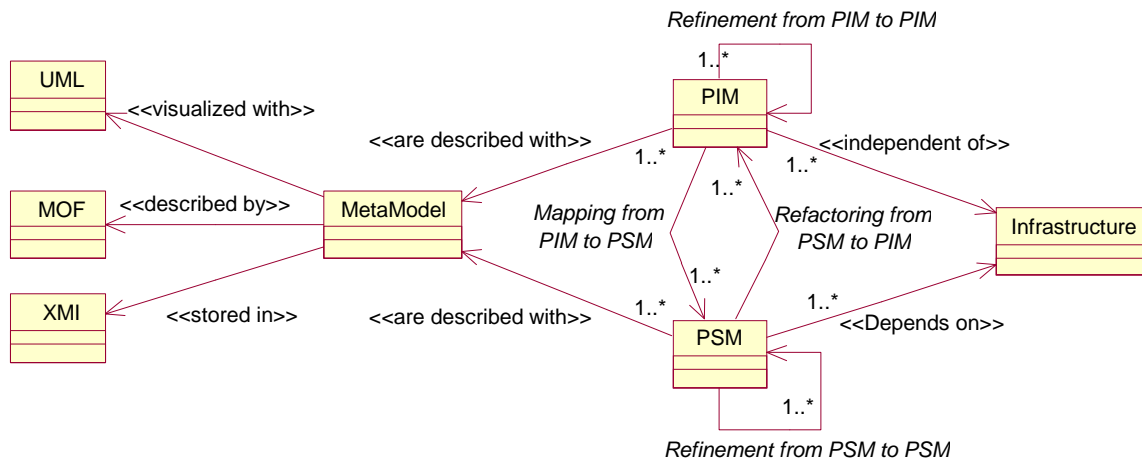


Figure 3: UML diagram summarizing the MDA development approach.

targeted platform's component model. It should be noted that if a platform does not support, or provides only partial support for a required service, then it may not be possible to meet the complete requirements of the PIM.

Figure 3 shows the relationship between the PIM and PSM and how some of the key technology standards, such as the Meta Object Facilities (MOF) and XML Metadata Interchange (XMI), are utilized by the MDA. As this figure demonstrates, a continual refinement process occurs at both the PIM and PSM levels, which are in turn mapped or refactored back to the PSM or PIM respectively to ensure that both models remain in step with each other.

Once the PIM and PSM have been completed, automated code-generation tools can be used to create the system's components. The PSM should contain the same level of detail as the coded application. This detail is stored in UML, using Object Constraint Languages (OCL) and Action Semantic Language to more accurately specify the internal behavior of a component. However, while it is the ambition of the MDA to completely model and automate the generation practice it is still some way to being fully realized. Currently, the most advanced MDA tools model only the behavioral aspects of a system for a given industry (arguably leaving the more complex considerations to the programmer); other tools take a more component-oriented approach and model just the container services [10]. Regardless of the degree of automated code generation, the use of MDA based tools has the potential to deliver a significant saving in development time by allowing the developer to concentrate on business logic concerns rather than integration logic. This ultimately leads to better quality code with reduced development risk. The entire MDA development process is shown in Figure 4.

2.3 Current state of MDA

While many of the concepts being advocated are not so different from those of CASE tools from the early to mid 1980s, the software industry as a whole has matured. As a result, unlike CASE tools that were proprietary in nature, the MDA is based on standard architectures, employing well-understood open standards like UML, MOF and XML. Beyond the technical issues, the MDA is generating strong support from the business sector, which recognizes the need for improved software process and tools in order to improve their return on software investment while managing technological change.

While the MDA is still a relatively new entrant with limited tools support, the future for this approach to software modeling and development looks promising. Not only is the MDA now the key technology focus from the Open Management Group (OMG), arguably the world's most influential standards body, but it has also received strong industry support from internationally recognized companies such as IBM Software, IONA Technologies, Lockheed Martin, Rational Software and Thales Research and Technology [10].

The next section will explore how the MDA can be applied to the areas of simulation and what work needs to be done to realign HLA to MDA.

3. Applying MDA to HLA

While the MDA currently has an enterprise focus, its goals and benefits are equally applicable to the Modelling and Simulation domain. This section will look at the benefits that the MDA could bring to the simulation domain, and what is required to bring HLA, and simulation in general, in line with the MDA methodology.

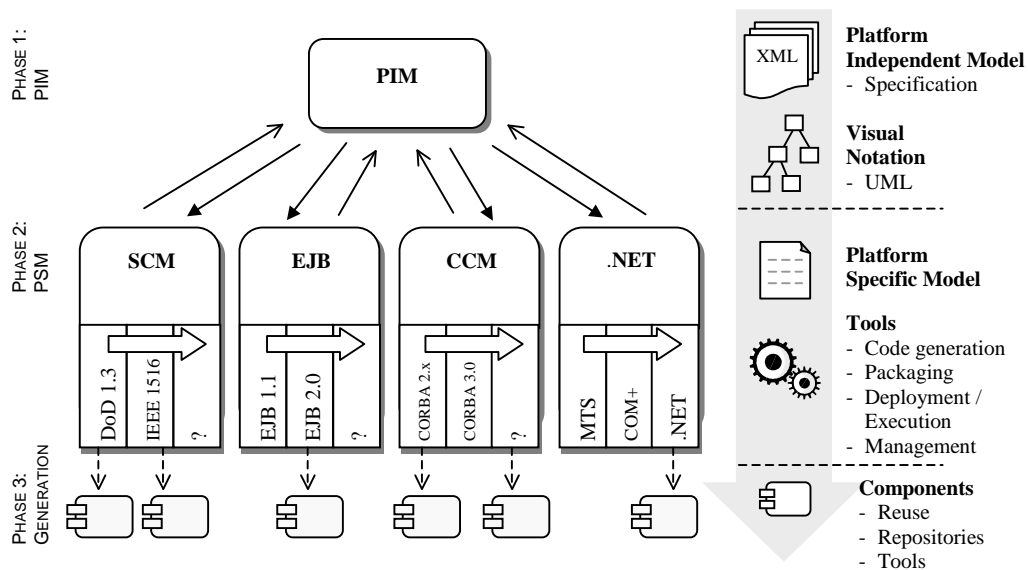


Figure 4: MDA Development Phases

3.1 Benefits of the MDA to simulation

In the previous section we outlined a number of the motivations behind the development of the MDA, both from a technical and business perspective. Before moving on to discuss some of the implementational issues in supporting MDA, it is worth summarizing these benefits again in context of simulation and in particular HLA:

1. The adoption of MDA would allow the HLA specification to be realigned with current industry initiatives and architectures. This has two important benefits. Firstly, it ensures that the HLA standard remains current, broadening the exposure and hopefully the uptake of HLA beyond the defense sector, while in the process avoiding another *Green Elephant* [1]. Secondly, it allows HLA developers to make use of new open standards and services, such as Web Services and security, within their simulations.
2. MDA promotes the adoption of a standardized component model, which in turn improves reuse, quality and interoperability, while reducing development effort, maintenance and costs.
3. MDA promotes the use of formal design methodologies and notation such as UML.
4. MDA moves the development focus away from the implementation/middleware concerns to the business or behavioral aspects of the simulation. Using MDA the model becomes the key design artifact of the simulation, promoting greater understanding between simulation stakeholders. Furthermore, the MDA process ensures that the behavioral model is not polluted by the integration concerns of the target platform.
5. MDA provides a mechanism to transition simulations over time as standards mature and change. A review

of the simulation arena points to a number of compatibility and integration issues between DIS, HLA 1.3 and IEEE 1516 standards. While work is currently progressing on addressing some of the RTI vendor interoperability issues [4], this work can only ever provide a short-term solution, as standards, platforms and services will continue to change.

6. The MDA approach provides an opportunity to support the whole development process through the application of unified tools [13].

The benefits derived from the application of the MDA have been well documented [1, 5, 2], and it would seem that HLA could benefit significantly from its adoption.

While the work required to develop an MDA based solution for HLA would be non-trivial, a number of the key requirements either already exist, or are well understood and documented. The following section will examine the three key requirements for accomplishing this task; namely the development of a *component model*, the specification of a *UML profile* for HLA, and finally the development of *unified tools* to support the MDA process in simulation.

3.2 Requirement 1: A component model for HLA

The MDA methodology is based around the idea that software is composed of components. During the modelling process components are described in generic terms at the PIM level and then transitioned to a given platform through a PSM. In turn, the PSM and any code generation tools then rely on the component model of the target platform to provide a standardized method for realizing the requested services in code.

While HLA does promote a component-oriented approach to federation development, there is currently no formal or ratified component model that standardizes the development of HLA components [2, 3].

Overview of CBD

Component-based development (CBD) is now a well-established methodology [17, 18, 19] and one of the cornerstones of the MDA approach. CBD describes a methodology where a system is broken into self-describing modules (Components) that describe their services (the underlying implementation) in the form of a standard *interface specification*. The goal of CBD is to increase reuse while reducing development and maintenance costs. Like the MDA, CBD strives to maintain a separation between integration and business logic concerns, thereby preventing the underlying middleware or integration requirements from polluting the business logic.

The design goals and philosophy of CBD also share many similarities with the goals of HLA federate development, including the promotion of reuse and interoperability. However, HLA currently has no standardized development approach to CBD, known as a *component model*, analogous to the CORBA Component Model (CCM) or the Enterprise Java Bean (EJB) container. As a result, there is no commonality in design and implementation between federates, or any formal way of separating and reusing the behavioral aspects of an HLA component. This results in an increase in development and maintenance costs, as well as impacting on the potential for reuse outside of the FOM for which the federate was built.

The Simulation Component Model

The Simulation Component Model (SCM) [2, 11, 12] is one potential candidate for the definition of a standardized model for component-based simulation development. This architecture employs a component model based on the OMG's CCM, and describes how the separation of integration logic and simulation behavior can be achieved for an HLA federate. This SCM development model also provides a commonality of design between federates, allowing them to access both HLA and extended CBD services (such as data transformation services between federates and the FOM) in a consistent manner.

A schematic of the SCM component architecture can be seen in Figure 5. In this architecture, a Federate is internally separated into *Behavior Logic* and *Integration Code*. The integration code manages the interactions between the behavior logic and the RTI, providing an RTI independent API to the behavior code, and a simulation independent API to the RTI services. This layer of abstraction between the RTI and the actual behavior logic

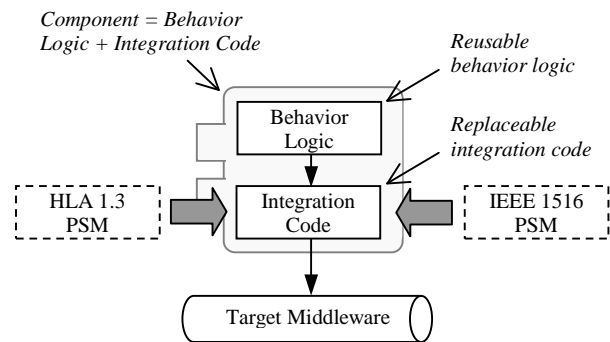


Figure 5: Simulation Component Model.

not only simplifies the development effort but also allows the integration code to be regenerated for various middleware layers (e.g. different PSMs for different vendor RTIs) without impacting upon the behavior logic.

The behavior logic encompasses the algorithmic description of the process under simulation; any simulation related services required by the behavior logic are accessed via the Integration Logic, rather than through direct interaction with the RTI (although direct RTI access remains an option). The use of these abstractions ensures that the Federate remains decoupled from a specific RTI vendor implementation. The Federate is then deployed in an RTI container.

Component Specification

Another important characteristic of a component is the ability to describe it via a *specification*, which is decomposed into a metadata *descriptor* of the component and an *interface* that describes, in a standard format, both what the component provides/publishes and uses/subscribes (a federate's SOM).

The component's implementation is a realization of the component interface; the collaborations defined in the interface are reflected in the component's implementation. For example, in the case of a *provided interface*, the component must have the required business logic for operations, and the appropriate accessors and mutators for each attribute. To expedite the development process the initial implementation can be automatically generated from the component specification, designed in UML, using code generation tools [3, 2, 13].

The component descriptor also contains metadata that specifies the requirements of the implementation. This includes the programming language, language specific dependencies (such as header and library files for C++), and operating system requirements (such as shared libraries). This enables multiple component implementations, targeted at various language/operating

system combinations, to co-exist in a single deployable unit.

Advantages of the SCM

The Simulation Component Model promotes a number of advantages:

- It provides a means of constructing a simulation from discrete self-describing components. While this is one of the goals of HLA, reuse is currently constrained by the FOM-centric nature of federation development, as well as the lack of a standardized component model.
- It provides the ability to reuse and extend a component's functionality without modifying the original implementation. This includes the ability to transform data between a federate's SOM and a federation's FOM in the integration code without affecting the behavior logic of the component (see Figure 6) [14]. In addition to transformations, other services (referred to as pervasive services by the MDA) such as security, transactions and persistence, can also be provided in the integration code.
- It improves the overall readability and quality of the component implementations through the application of design patterns to promote best practice, resulting in reduced code complexity and greater knowledge transfer [18]
- It provides appropriate standards for process and data, which encourages the development of tools that assist developers from design through to execution [12, 13]. This includes mappings that define how to go from a specification (PIM) to a programming language (PSM).
- It facilitates interoperability within a heterogeneous environment. This includes interoperability between

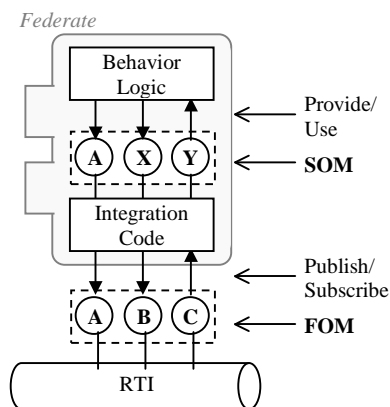


Figure 6: A Component-Based federate, with loose coupling to the simulation's FOM and tight coupling to the component's own SOM.

simulation specifications such as DIS and HLA, as well as providing broad language and platform support for components.

The SCM demonstrates the applicability of CBD to HLA, while laying the groundwork for the development of a complete component model specification for federate development.

3.3 Requirement 2: Developing a UML Profile

In order to accommodate the broadest possible user-base, the MDA provides a generic mechanism for extending the base UML model through the development of domain specific *UML profiles*. A UML profile provides a collection of additional Stereotypes and Tagged values that can be applied to elements, attributes, methods, links and other UML notations in order to facilitate the construction of a complete model for a particular domain. Each of the UML extensions must also be understood and modelled at the PIM and PSM level, and the actual services described by the new stereotypes must be supported by the target component-model.

While the development of a UML profile represents a significant effort, which is well beyond the scope of this paper, this task is considerably simplified by the maturity of the HLA standard. For example, a cursory examination of the HLA standard quickly starts to generate a reasonably complete superset of the main stereotypes and tags that would need to be defined to fully describe an HLA simulation. These include:

- Object and interaction classes
- Publication and subscription of data
- Synchronization
- Time management
- Object management
- Dynamic data management
- Persistence
- Wiring rules (e.g. byte order)

In addition to supporting the pervasive service described by the MDA, the SCM could be extended to support additional simulation domain services through a UML profile, including:

- Inter-federate data transformations
- Dead reckoning
- Scenario management
- Unit conversations

Possible extension or specializations of the UML notation [14] should also be considered as part of the UML profile for simulation. For example, as shown in Figure 7, while class diagrams can be extended through stereotypes and

tags to indicate the publication and subscription of object classes between federates, specialized UML diagrams could be employed to help manage complexity and facilitate better communication between developers.

So while the development of a UML profile, like the SCM, would require community involvement and agreement, possibly through the OMG, the actual mechanics and the scope of the task is reasonably well understood.

3.4 Requirement 3: Unified tools set

The previous sections introduced the SCM and how the component specification, in conjunction with a UML profile, can be used to capture the complete specification of a system through UML models. While this approach facilitates increased repeatability, communication, reuse and interoperability, it does not simplify the job of the developer, as he or she must still develop, deploy and execute the simulation manually.

To address this issue, the MDA approach encourages the application of tools [6, 10] for modelling the design of a system, and for automating repetitive development tasks, such as the generation of component code. These tools lead to increased productivity and improved development consistency and quality.

Areas of tools support

Tools can be applied across the whole MDA development life cycle to provide a unified approach to simulation development. These tools can be summarized into the following three categories:

- **Design:** Pivotal to the MDA development approach are design tools to assist and simplify the modeling and specification of the PIM and PSM. The design process is centered on the design and management of a system's UML models and their related metadata, and ideally encompasses all aspects of the system from business behavior to component deployment.
- **Generation:** One of the most compelling applications of tools is in the area of code-generation. Automated generation engines that use code templates can be employed to transform the PIM and PSM specifications into the bulk of the boilerplate code required by a component.
- **Deployment and Execution:** One of the characteristics of a component is the ability to package all of its constituent files, including its interface and metadata description, into a self-contained and self-describing package. Tools can then be used to package, deploy, execute and manage a system's components across a network.

Auxiliary tools such as version control, component repositories, requirements gathering and run-time performance metrics software could also be incorporated into the tools collection.

While this is a reasonably extensive list of tools, many of these already exist in one form or another, including the first iteration of an MDA based Integrated Development Environment (IDE) for HLA from Calytrix Technologies [12, 15]. While there will be work required from both open-source and commercial vendors, the development of a unified tools environment for HLA is an achievable goal.

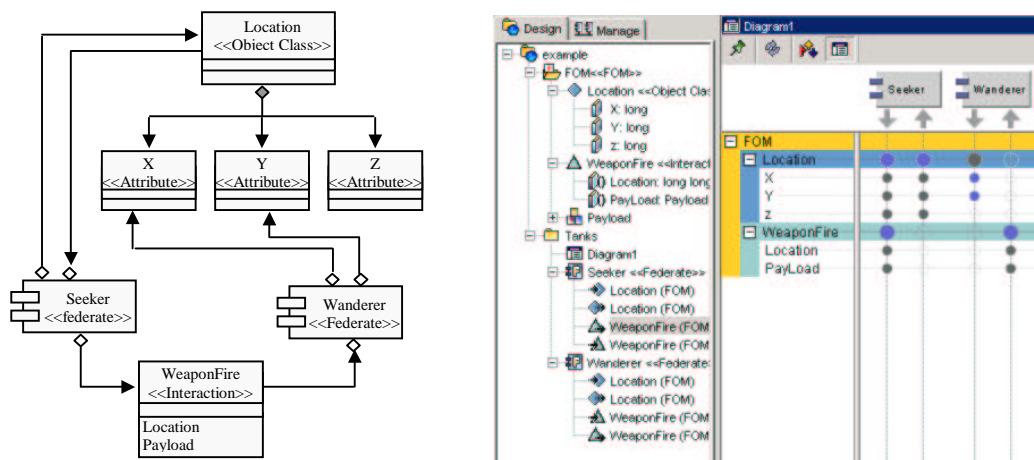


Figure 7: Specialized UML diagrams can be employed to reduce complexity.

Insuring interoperability between tools

The MDA also provides a common mechanism for storing and exchanging models and metadata between tools through the application of XML Metadata Interchange (XMI) and the Meta-Object Facility (MOF). MOF is the standard modeling and interchange construct used by MDA tools for storing models, including UML models. This common foundation provides the basis for model and metamodel interchange, ensuring interoperability between tools. XMI defines an XML-based interchange format for UML models by standardizing a number of XML document formats and Document Type Definitions (DTD). In so doing, it provides a mechanism for mapping UML models into XML. XMI, which unites modeling (UML), metadata (MOF and XML) and domain specifics (UML profiles), is a crucial element of the MDA approach. It is essential that any tool set developed supports MOF and XMI, as well as a number of other open standards adopted by MDA [16].

4. Conclusion

While still relatively new, the Model Driven Architecture is rapidly gaining support from both technical and business groups alike. Building on this momentum, this paper has attempted to provide the reader with some background to the motivations driving the MDA, as well as an overview of how the MDA development process works. This paper also describes how the application of MDA could benefit HLA, as well as summarizing the key technical requirements needed to realign HLA to integrate with MDA.

Significant work remains at the standards and implementation level. UML notations, UML profiles, a component model and tools must all be developed if HLA is to align itself with the goals of MDA. However, HLA is technically well positioned to leverage the advantages of MDA. All that is required is a considered effort on the behalf of the HLA community to grasp the opportunity afforded it.

References

- [1] Tolk, A., "Avoiding another Green Elephant - A Proposal for the Next Generation HLA based of the Model Driven Architecture", Proceedings of the 2002 Fall Simulation Interoperability Workshop (SISO Fall 2002). Paper ID 02F-SIW-004, November 2002.
- [2] Radeski, A., Parr, S. "Towards a Simulation Component Model for HLA". Proceedings of the 2002 Fall Simulation Interoperability Workshop (SISO Fall 2002). Paper ID 02F-SIW-079, November 2002.
- [3] Cox, K., "A Framework-based Approach to HLA Federate Development". Proceedings of the 1998 Spring Simulation Interoperability Workshop (SISO Fall 1998). Paper ID 98F-SIW-181, 1998
- [4] Dynamic Link Compatible HLA Working Group, Simulation Interoperability Standards Organization's (SISO), <http://www.sisostds.org>, 2002
- [5] Open Management Group, "Model Driven Architecture", <http://www.omg.org/mda/>, 2001
- [6] Calytrix Technologies, "Calytrix SIMplicity - an MDA approach to distributed simulation", <http://www.omg.org/mda/committed-products.htm>, 2002
- [7] Object Management Group, "Updated CCM Specification", <http://www.omg.org>, 2001
- [8] "Extensible Modeling and Simulation Framework (XMSF)", <http://www.movesinstitute.org/xmsf/xmsf.html>
- [9] Siegel, J., "Making the Case: OMG's Model Driven Architecture", Software Development Times (www.sdtimes.com), 15 October 2001
- [10] Open Management Group, "OMG Members and Industry Analysts Support the MDA", http://www.omg.org/mda/mda_files/Member_and_Analyst_Quotes.pdf, 2001
- [11] Radeski, A., Parr, S., Keith-Magee, R., & Wharington, J. "Component-Based Development Extensions to HLA". Proceedings of the 2002 Spring Simulation Interoperability Workshop (SISO Spring 2002). Paper ID 02S-SIW-046, March 2002.
- [12] Calytrix Technologies. "Welcome to SIMplicity", <http://www.simplicity.calytrix.com>, 2002
- [13] Parr, S., Radeski, A., & Whitney, R. "The Application of Tools Support In HLA". Proceedings of the Simulation Technology and Training Conference 2002 (SimTecT 2002). Paper ID 26, 2002
- [14] Keith-Magee, K., Parr, S. "Visualising Distributed Simulation Design And Deployment", Proceedings of the Interservice/Industry, Simulation and Education Conference (IITSEC) 2002, Paper ID 258, 2002
- [15] Open Management Group, "Committed Companies and Their Products", <http://www.omg.org/mda/committed-products.htm>, 2002
- [16] Open Management Group's Architecture Board, "Model Driven Architecture", Document number ormsc/2001-12-01, <http://www.omg.org/mda/>, December 2001
- [17] Syzperski, "Component Software – Beyond Object Oriented Programming". Addison Wesley, 1998

- [18] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. Elements of Reusable Object-Oriented Software. Addison Wesley, 2000.
- [19] Heineman, G., & Councill, W. "Component-Based Software Engineering". Addison Wesley.

Author Biographies

SHAWN PARR is the co-founder and Chief Technical Officer at Calytrix Technologies, an Australian based research and development company specializing in HLA

simulations, the Model Driven Architecture and component-based design. Shawn has been working in the IT industry for over 10 years and holds a Bachelor of Science degree and a research based Masters degree.

DR RUSSELL KEITH-MAGEE is a Software Engineer at Calytrix Technologies with a research background in both Physics and Computing. His doctoral thesis involved an analysis of biologically motivated models of machine learning and development. He also holds a Bachelor of Science with Honours in Computer Science and a Bachelor of Science in Physics.