Applying Model-Driven Development Techniques to the Development of Search and Rescue Systems

Huy Pham
Dept. Comp & Info Science
University of Guelph
Guelph, Ontario, Canada
hpham@uoguelph.ca

Alexander Ferworn
Dept. of Computer Science
Ryerson University
Toronto, Ontario, Canada
aferworn@scs.ryerson.ca

Qusay H. Mahmoud
Dept. Comp & Info Science
University of Guelph
Guelph, Ontario, Canada
qmahmoud@cis.uoguelph.ca

Alireza Sadeghian
Dept. of Computer Science
Ryerson University
Toronto, Ontario, Canada
asadeghi@ryerson.ca

Abstract - This paper describes our work of applying modern software engineering methodologies such as Model-Driven Development and Product-Line Engineering to the development and maintenance of technical supporting systems in the domain of Search and Rescue. We propose an extensible, domain-specific, graphical modeling language and toolset that allow software developers of Search and Rescue systems to rapidly compose and generate their applications from a set of pre-defined graphical primitives that represent the common building blocks of SAR applications. A case study for the proposed toolset is also presented.

Keywords: Search and Rescue Software Development, Model-Driven Development, MDD, SAR, USAR.

1 Introduction

Hazardous material, vast areas of destruction and the need to find and rescue people quickly have made emergency responders turn to advanced technological support systems, such as rescue robots [1] and aerial surveillance systems [2]. From a System-of-Systems Engineering point of view, these systems are often complex and involve a number of highly coordinated sub-systems and devices. Developing and maintaining software for such systems is a challenging task for a number of reasons [3]. First, each subsystem can have a different operational model and each device in those subsystems can have a different execution or programming model. This makes software reuse across different devices and sub-systems difficult. Secondly, many of the devices have special resource constraints, such as limited memory, storage, battery, and computational power. This makes middleware solutions, which are usually used to overcome the difficulties associated with the large variety in execution and programming models, impossible in some cases, simply because some devices do not have the capabilities required to host a middleware platform. As a consequence, the “Write once, run anywhere” paradigm may not be applicable, and multiple versions of the same software, one for each target execution platform, may need to be implemented and separately maintained. Lastly, recent technological capabilities have tended to change quickly in this domain, and new devices and subsystems frequently need to be added, removed or replaced. This fact, compounded with the need to maintain multiple versions of the same software, makes software maintenance in this domain a potentially costly process.

Model-Driven Development (MDD) [4] and Product-Line Engineering (PLE) [5] are relatively new software engineering approaches that promise to overcome these difficulties and significantly increase the productivity of software development and maintenance. This paper describes our work in applying MDD and PLE techniques to the development and maintenance of SAR software systems. It is organized as follows. Section 2 introduces the concepts of MDD, PLE and Domain-Specific Languages. Section 3 describes the common building blocks of SAR software systems, together with our proposed graphical modeling language, which encapsulates these common building blocks as pre-defined graphical modeling primitives, and allows software developers in the domain of SAR to quickly compose and generate their applications by dragging those graphical primitives onto a design surface. Section 4 provides a case study in which we use this modeling language to develop various components of a real-world Search and Rescue software system. Finally, section 5 contains our conclusions and future work.

2 Background

2.1 Model-Driven Development

Since the introduction of third generation programming languages like C, C++ and Java, software developers have seen a big breakthrough in productivity. Instead of having to construct their applications from primitives like load, store and jump, developers are able to express their logic using higher and more natural programming constructs like loops, conditionals, etc., and then call upon a compiler to generate the machine-level code. Also, using these languages, the task of porting software to a new execution platform when it comes along involves only a recompile, as opposed to a rewrite or a redesign, of the programs in the system.

These benefits came from the basic idea that, by raising the abstraction level of the language used by developers, and by automating the process of transforming code from the raised abstraction level to the target (lower) abstraction level, productivity in both software
development and maintenance can be significantly improved.

Figure 1. Software development using 3rd gen languages

Model-Driven Development is a software engineering approach that aims to push this idea one step further. It proposes a software development methodology in which software is developed not by writing code directly in implementation languages like Java or C#, but by constructing high level models that can be transformed into code by automated transformation engines and code generators.

Figure 2. MDD software development

Like 3rd generation languages, this approach offers two major advantages. The first is that, since high-level modeling concepts, as compared to those found in implementation languages, are much closer to the real concepts in the problem domain, software specification, understanding, and development are much easier with models. The second major advantage, which is captured in MDD’s “Model once, generate anywhere” slogan, is that, since the concepts used in the models are less bound to the underlying implementation technology, software is less susceptible to technological change. This makes software maintenance easier and more economical.

2.2 Product-Line Engineering

While MDD aims to increase the productivity in software development and maintenance by raising the abstraction level and shifting the focus from coding to model construction, Product-Line Engineering seeks to accomplish the same goal by exploiting the commonalities between similar systems of the same problem domain. In this approach, instead of building each system individually like before, developers would first develop a set of general and reusable development assets such as architectures, components, models, etc. that can be shared among all the systems of a particular family. Once these development assets have become available, they can be used to quickly create different variants of the target family in an efficient and systematic manner [6, 7].

In light of Model-Driven Development, the set of reusable assets of a system product line can be extended to include not only architectures, components, models, etc. but also modeling languages, modeling tools, and transformation engines and code generators.

2.3 Domain-Specific Languages

Modeling languages come in different shapes and sizes. Some languages are more general than others. Like programming languages, a modeling language’s scope of application is proportional to its generality, while its expressive power in a given problem domain is proportional to its specificity to that particular domain [8-10]. It has been argued [11] that general modeling languages like Unified Modeling Language (UML), while applicable to a large number of domains, are not well-suited for the purpose of machine understanding and processing, one of the primary premises of MDD. Furthermore, in order to realize the MDD vision, modeling languages would—besides having to have a well defined syntax and semantics—have to be specific to their target domains, so that they can be expressive and powerful enough to be useful for the purpose of solving problems within that particular domain. For this reason, there has been a growing trend in using domain-specific languages (DSLs), as opposed to general languages like UML, as the modeling languages in MDD.

3 A Domain-Specific Language for SAR Software Development

In this section, we propose an extensible, domain-specific, modeling language that is targeted to the domain of SAR software development. From a Model-Driven Development’s point of view, this language, together with its supporting components, i.e. an accompanying graphical designer and a set of code generation templates, can be considered to be a model-driven development tool, because it allows software developers to quickly compose and generate softwares for SAR systems from a set of pre-defined graphical primitives that represent the common building blocks of SAR applications. From the Product-Line Engineering’s point of view, it can be considered to be a set of shareable and reusable development assets for the development for applications in the SAR family of systems. The language is extensible in the sense that new primitives representing new building blocks can be added at anytime.

We developed this DSL and its supporting components using the DSL Tools framework [12], a Visual Studio-based experimental meta-modeling framework for domain specific languages, provided by Microsoft as part of its MDD initiative that allows system designers and domain experts working in different domains to design and specify modeling languages for their domains.

3.1 Basis building blocks of SAR software systems

One thing that we have come to note from our experience with designing and developing software for rescue robotic systems is that they usually involve capturing, transmission and processing of GPS, Video and Audio signals as well as other sensor data. Because these
features are shared among several remote SAR applications, we realized that one way to increase our productivity is to devise a systematic way of sharing and reusing the software components that implement these features. Furthermore, we realized that productivity of software development and maintenance in this domain can be greatly improved by a framework that could 1) facilitate easy component reuse and generation, and 2) allow good software designs and practices to be encoded in a coherent manner. So we set out to apply MDD techniques and build a domain-specific modeling language that would allow us to do just that.

3.2 Rolling up a domain-specific modeling language

Recent attempts to characterize and create descriptions for a small subset of SAR technology through a standardized ontology would indicate that there is much room for modeling within the domain [14].

Our first step in building a modeling language was to identify, for the purpose of reuse, a set of basic components that can be shared by all SAR supporting systems within our limited use of the term. For instance, we have divided (i.e. componentized) our systems, by defining the necessary interaction interfaces, into components such as GPS Signal Transmitter, GPS Signal Receiver, which handle the transmitting and receiving of GPS signal over the network, and GPS Signal Recorder, GPS Signal Visualizer to handle the recording and displaying received GPS signals from various sources.

Our next step is to generalize these components to make them reusable in a wide range of contexts. We did this by first identifying and parametrizing all the variability points of each identified component. These variability points can be thought of as configuration variables that need to be specified when the component are deployed. One example for this is the TCP port number at which the GPS Signal Receiver, which is implemented as a network server, listens on. In order to allow this component to be reused in a wide range of contexts, this variability point must be parametrized and treated as a configuration variable of the component.

Once all the components have been generalized, we wrap them into a modeling framework as a set of graphical modeling primitives and a set of accompanying code generation templates. The graphical primitives represent the generalized components, with their configuration variables represented as the primitives’ properties (Figure 4 below), and the code generation templates are used to transform these primitives into codes that represent the actual implementation of the components.

Figure 3 shows our resulting graphical modeling environment, hosted inside Microsoft’s Visual Studio. The graphical primitives representing the basic building blocks of SAR software systems and their possible relationships are shown as items in the toolbox on the left.

Figure 4. Viewing and specifying the variability points of the GPS Receiver component
4 A Case Study

To demonstrate the use of our DSL, we have applied it to the development of some software components of an Urban Search and Rescue (USAR) system development project. This project, called the Canine Augmentation Technology (CAT) project [13], is a part of a cooperative effort between our research group and the Provincial Emergency Response Team (PERT) of the Ontario Provincial Police (OPP). Its primary intent is to provide useful technological component augmentation to the force’s USAR Canine teams in support of USAR operations where direct interaction is precluded, for example in certain rubble search scenarios or in the rapid systematic search of distributed partially collapsed or otherwise unsafe structures. The work stems from the premise that it is unlikely that low-level autonomous mobility systems will be available in the foreseeable future that can match or surpass the ability of canines to move over rubble quickly.

![Figure 5](image)

**Figure 5.** An OPP Officer and his CAT-enabled dog on a rubble pile

4.1 Overall Architecture of the CAT project

Figure 6 illustrates the overall architecture of the CAT system. As can be seen, CAT consists of at least three different sub-systems: One *USAR Dog* system for each canine involved in the SAR operation, one *Handler* system for each canine handler involved, and one *Observer* system for the central observer/coordinate. The Dog system is mounted on a custom harness on the dog. Its function is to gather, encode, and transmit sensed data such as the dog’s GPS location, video and audio, etc., to both the handler and observers. The Handler system is carried by the dog’s handler. Its function is to receive data from the dog, gather and transmit the handler’s location data to the observer, and transmit voice commands to the dog. Finally, the Observer system gathers all data transmitted by all the dogs and theirs handlers, and presents them to the observer in a coherent manner.

![Figure 6](image)

**Figure 6.** Overall Architecture of the CAT system

4.2 Building CAT’s GPS and Video components

We used the proposed modeling language to model and construct the GPS and Video software components for each of the three CAT’s subsystems. Figure 7 shows the model of these components for the Observer subsystem. At the bottom of the diagram are the graphical primitives representing the *GPS Signal Receiver* and the *Video Signal Receiver* components. Each of these primitives will be transformed, by the accompanying code generation templates, into a multi-threaded TCP server, with each thread handling one source of signal.

For example, the *GPS Receiver* will be transformed into a network server that spawns, for each GPS signal transmitter (i.e., a dog or a handler), a separate thread. This thread reads the signal from the transmitter over the network, formats it into proper data structure, and then passes it up to the *GPS Signal Broker* component. This component receives data from both the *GPS Receiver* and the *GPS Device Reader*, which reads data from a GPS device mounted on the Observer system, and broadcasts them as events to all interested event listeners, which are the *GPS Plotter* and *GPS Recorder* in this case. These two components, upon receiving an event, will read the formatted signal, classify it according to the source ID, and then either store it to permanent storage, in case of the recorder, or plot it on the Observer’s screen, in case of the plotter.
The GUI Window Screen components at the top of the diagram will be transformed into two simple Windows forms that host the GUI elements associated with the four components beneath it (the GPS Signal Plotter, Recorder, the Video Signal Displayer and Recorder components). Figure 8 below gives a screenshot of the generated window containing the GPS Plotter and the GPS Recorder.

![Diagram of the Observer’s GPS and Video components]

Figure 7. Model of the Observer’s GPS and Video components

![Screenshot of the generated application]

Figure 8. A screen shot of the generated application

5 Conclusions and Future Work

We have presented a domain specific modeling language for the a subdomain of SAR and USAR software development. This language, together with its supporting tools such as the graphical editor and the set of supplied code generation templates, provides SAR software developers with the ability to quickly composing and generating their applications from a set of supplied graphical modeling primitives that represent the common building blocks of SAR applications. We argue that this extensible language would help system developers in this domain to significantly increase not only their productivity but also the quality of their applications, because it encapsulates all the good software designs and practices that we have learned from our experience working in this domain. To demonstrate its use, we have applied the language to the development of two sub-components of a real-world SAR support system.

We intend to extend this language to include a more comprehensive set of SAR software building blocks. We are also planning to work on a model transformation engine that would allow more abstract and complex concepts in the domain to be modeled and then transformed into more concrete and fine-grained components, before code generation can be done with text templating.

References

