Enhancing Distributed Simulation Systems by Utilizing Component-based Technologies

Baharak G. Fard, Jari Ala-Kurikka, Andreas Kvarnström, Ivica Crnkovic

Mälardalen University, Department of Computer Engineering, 721 23 Västerås, Sweden,
bgd99001@student.mdh.se, jaa99001@student.mdh.se, andreas.kvarnstrom@mdh.se
Ivica.Crnkovic@mdh.se

Abstract: Simulations in today's industry are becoming increasingly complex and are requiring more and more computing power to gain higher efficiency. One way of solving this is by adding computing power to the machine performing the simulation. However, this approach is not only costly but also in some cases requires time-consuming re-programming of the simulation environment to fully utilize the increased performance. A solution to this is to divide the problem into different subtasks and distribute them on several machines running possibly different simulators. Although this at first glance seems to be an appealing solution, these types of systems tend to introduce new obstacles. In order to minimize costs, these distributed simulation environments often need to include many already existing systems. The simulators are also in many cases closed environments, which are difficult to communicate with due to their varying data format, communication protocol and data exchange models. For such heterogeneous systems it is a challenge to obtain a seamless integration, in particular if there are requirements for system expansion either by integrating new simulators or introducing new simulations. This paper describes these challenges and a middleware concept that meets them. Further it gives an example the middleware component "the DOTS Middleware Model (DMM)" that has been developed and that meets the requirements and challenges discussed.

1 Introduction

Trying to establish integrated simulation environments including many different simulators and platforms automatically lead to problems when interoperation between the systems is considered; there are no pre-defined solutions for integrating multiple simulation environments that differ in many factors.

The DOTS EU project (Flexible and eco-efficient paper production through Dynamic Optimization of operational Tasks and Scenarios) is collaboration between many stakeholders in different countries that addresses the situation mentioned above. The main objective in the project is to use a number of pre-existing simulators in collaboration with a recently developed optimizer. Optimizations are performed with respect to different criteria (price, energy consumption, etc.) to achieve more efficient paper production. A toolset for dynamic optimization of process actions based on the underlying dynamic process simulators will be developed. Even though the project mainly focuses on the paper industry, the approach itself and many of the results can be generalized to suit other situations.

The aim of this paper is to introduce and discuss a component-based and middleware approach for integration of different types of systems. The advantages and disadvantages of such approach are discussed and an example of a middleware component that enables interoperation between the systems (simulators) is demonstrated.

We start in section 2 by examining the different challenges that arise in the development of distributed simulation systems. Section 3 deals with possible approaches to achieve interoperation between the systems, while section 4 examines the DOTS approach in more detail. Section 5 provides a use case of the latter model and the paper completes with a conclusion and future works.

2 Challenges of Simulator Interoperability

In order to optimize a particular process different optimization models can be used. In cases of complex processes, only analytic mathematical models cannot be used, but values of some variables can be provided by simulators. For different type of processes exist different simulators, and for complex processes, interoperation between different simulators is required. A total solution in such cases is a system that may consist of several simulators and of one or several optimizers. The main advantage of such approach is a possibility to build solutions for optimizing complex process by reusing optimizers and simulators as already existing components. For a seamless work, a seamless integration between these components is required. From the interoperability point of view two main aspects must be considered: Run-time behavior (performance and availability, i.e. the ability of the systems to provide the results in expected time frames), and life time characteristics (modifiability and interoperability, i.e. ability to easy improve or add new functions of a systems by adding new components-simulators and protocols of information exchange).

The system should have support for different types of simulators with different interfaces and data formats. Different platforms and operating systems also pose demands on the system, mainly concerning the communication protocol used for exchanging data. Furthermore, the countries and companies involved have their own specific requirements that need to be integrated into the system. Efficiency and security issues are other types of challenges that the system must be able to manage.

Since it is often necessary to add new types of data and features to such systems, it has to be open for modifications and be fairly easy to update; it must be flexible and extensible enough so that changes can be made easily without having to make extensive alterations to the overall structure of the system. It must also provide an integrated environment that makes combining different simulators possible.

The important point to keep in mind here is that the interoperability issues should not increase the complexity of the entire system. In addition, information and services provided should remain simple, standardized and efficient.

3 Possible Interoperation Solutions

Interoperability and integration is one of the main problems of system development, in many engineering domains [1]. There are different integration approaches, none of them being able successfully meet requirements outline in the previous section [1][2].

In general, there are three different ways to achieve interoperability:

- **Offline translation of data using temporary files and off-line filters:** This approach is quite ineffective and impractical; identifying and introducing manual procedures for data exchange is in practice too time-consuming and inefficient.

- **Full integration using a common engineering database:** In this case, there is a need for developing a common information model and a package gathering the functionality of all simulators involved using the same structures and data, a common API and a common user interface. However, the lack of common and standardized data types for the simulators makes this model unfeasible. Furthermore, adding new simulators would be rather complicated, as extensive changes must be made in large parts of the system.

- **Loose integration keeping isolated application and implemented interpolation "bridges":** Since each simulator in the system has its own API, every new simulator

introduced in the system requires implementing new interoperability functions. Component-based technologies, such as .NET [3][4] provide many standardized interoperability features, which radically reduce the distributed development efforts by for example providing distributed debugging features.

Creating adapters as middleware components that uses APIs of the simulators and optimizers makes is possible to achieve a better interoperability than filter-based and significantly less expensive that data-based solution. A disadvantage of this approach is a number of adapters that increases with the increase of the number of the nodes; For communication between n-nodes, n(n-1)/2 adapters must be implemented.

By creating a middleware application developing a common meta translation model and implementing translation between specific and common model and at the same time utilizing component-based technologies, a loose integration can be achieved. For this reason it is more convenient that a common meta-adapter is specified which connects all the components. The second approach has obvious advantages. When adding a new simulator, a new adapter between this simulator and the meta adapter must be created, not adapters between all possible connections to this simulator (see Figure 1).

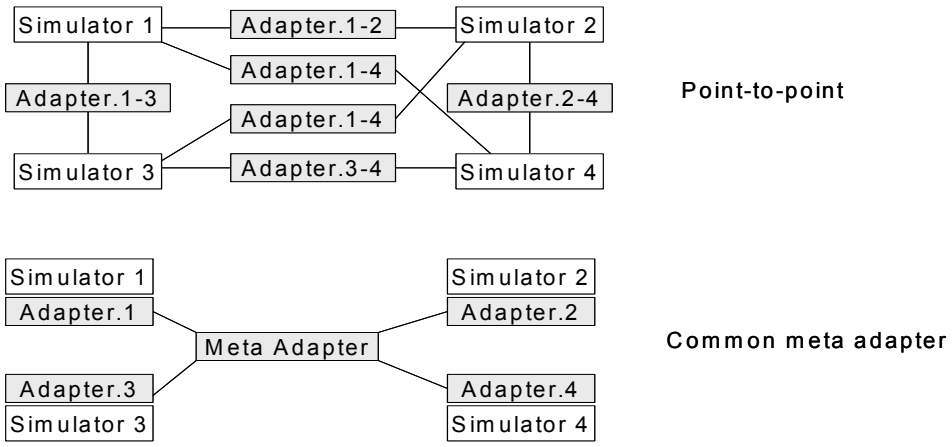

Figure 1. Point-to-point adapters and common mate adapter solutions

4 The DOTS Middleware Model

Following the reasoning described in the previous sections it was decided to specify a common meta interoperation model using a component-based technology and develop a middleware application (the DOTS Middleware Component). The DOTS Middleware components) uses a 3-tiers distributed architecture, dividing the system into three main layers in order to make it easier to modify and extend, when adding new simulators or data. The three layers together with their included nodes are described in Figure 2.

The *Optimizer Layer* contains one or several optimizer nodes, each including an optimizer and its corresponding adapter application. Optimizers communicate with simulators through the middleware components. In the concrete cases considering so far, the relation between an optimizer and simulators are master-slave. Communication between optimizers has not be considered, although the DOTS middleware component enables it.

In the same manner, the *Simulation Layer* includes one or several simulator nodes containing a simulator application and a matching adapter.

The key intercommunication function is performed in the *Middleware Layer*: handling the communication between the other two layers by using the adapters on each side. The *Middleware Component* is implemented as a .NET Web Service, using XML as data format and SOAP as communication protocol. All adapters in the system must also handle this format and protocol. More information concerning .NET, XML and SOAP can be found in [3]-[8].

In addition to the Middleware Component, the Middleware layer also includes a database. Database includes the intermediate results provided by simulators (and required by the optimizer). There are several reasons for storing results in the database: A simulation process may be time consuming and if the same process is repeated, a result that has been provided from a simulator before can directly be used. Similarly, the results may be used if a simulator is not running.

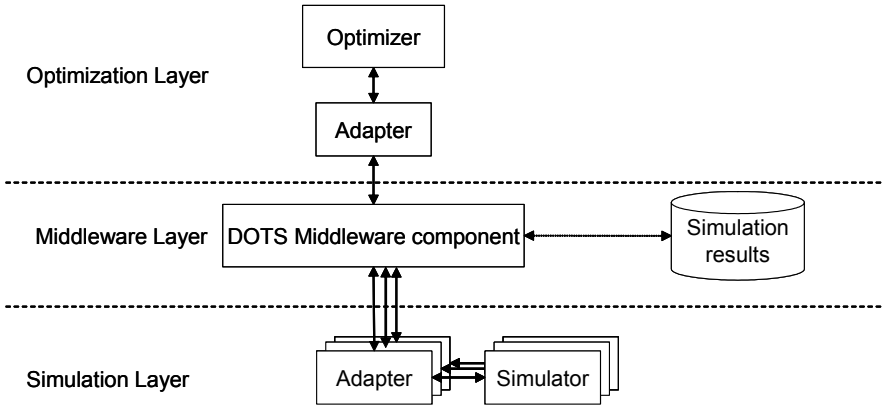

Figure 2. Overall system architecture

4.1 Benefits of the model

The distributed simulation environment is managed by utilizing the .NET component technology in conjunction with adapter applications. The included features in .NET, mainly the pre-defined communication protocol and data format, are used by the Middleware Component (MC). .NET provides services to calling applications via the Internet, no matter where they physically reside or which platform they use, as long as they support .NET. The second part of the solution lies in the adapter applications; they handle all communication between their respective simulator or optimizer and MC. All this is managed transparently to the end user.

Flexibility and scalability is mainly achieved through the diversity allowed in the participating node adapters. As long as they conform to the common communication protocol and data format, they are usable in the system. If the adapters in addition are developed in the .NET platform, also other advantages, such as language interoperability, are achieved.

The DOTS middleware model is extensible in the sense that new data types and formats can be easily added to the system. Only one part, the MC, needs modification, yet the change become available to the whole system. The same applies to the process of adding new simulators. Since MC together with the adapters handle all communication and data conversion, the users need not bother conversions of data between different simulators or optimizers, but can use the formats that are defined by the local simulators.

Many of these benefits are due to the extensibility and flexibility offered by XML and SOAP. Moreover, security problems such as firewalls are automatically avoided, since one of the protocols included in SOAP is HTTP.

4.2 Limitations of the model

Most drawbacks with the model occur when the system is deployed very first time. The proposed solution for the distributed environment imposes development of all adapters for each new node connected to MC.

The same goes for the efficiency problem; many simulations need to be carried out before the performance benefits can be fully achieved. Distributed simulation via Internet may call for extensive data exchange and may introduce overhead to the time spent on each simulation before many simulation results have been stored.

True platform independence has not yet been fully reached, as for the full interoperability a .NET complied platform is required. For simulators running on platform that not include .NET technology (for example some UNIX platform) require additional way of communication. It that cases a client application on the simulator site must be developed that can start the simulation process and pass it to MC.

5 Case Study: Optimization of Paper Machine Grade Changes

When creating optimization of a new process, one part, related to the MC, is to identify the data that will be passed between the optimizer and the simulator(s). In this section we illustrated the definition process by a case study.

In a paper mill, a single paper machine is often used to produce many different grades of paper according to customer quality and quantity requirements and delivery deadlines. Efficient scheduling of grade changes ensures that customer requirements are met, and that the paper machine produces a minimum amount of off grade paper in the process of changing from one grade to another. Off grade paper can in many cases not be directly used, but must be reprocessed in order to reach sufficient quality.

Paper grades are defined by various properties, such as ash and moisture content.

When the production of a paper machine is changed to another grade, a certain amount of time is needed to adjust the machine to produce the new grade. Since the paper produced during this time often is of no direct use and the production value of the machine is close to zero, there is a need for minimizing this period of time. This also defines the optimization problem used in this case study.

This case can be solved using an optimizer. However, to reach a feasible solution for a particular paper machine, the solutions need to be verified using simulations, taking into account the specifics of the machines involved.

5.1 Simulation Process

Let us examine a grade change problem closer with respect to the variables involved. In this scenario, three different aspects are considered:

- **Grammage (GM)** – The mass of the paper per square meter (g/m^2)
- **Ash Content (AC)** – The percentage of inorganics in the paper (%); related to the amount of chemical additives such as fillers
- **Moisture Content (MC)** – The percentage of moisture in the paper (%)

Grammage depends on the end use of the paper and can vary from low values for tissue grades to high values for board. Chemical additives can be used to modify the optical, strength, or surface characteristics of the paper. The moisture content affects both the behavior of the paper on the machine, and some properties of the finished product. All three aspects are modeled as different quality tubes with constraints.

5.1.1 Input and Output Parameters

Here the input parameters of a more specific instance of the above-mentioned case are defined. The paper machine is shifting production from the one paper grade described to another. GM is changed from 50 g/m^2 to 70 g/m^2, AC from 10 % to 15 % and MC is only altered during the grade change from 5% to 0%.

The variables in both tables are represented by time series defining the values of each variable at every point in time, which can be depicted as in 0. The variation times can also be seen in the figure. A constant time step, 10 s, is used.

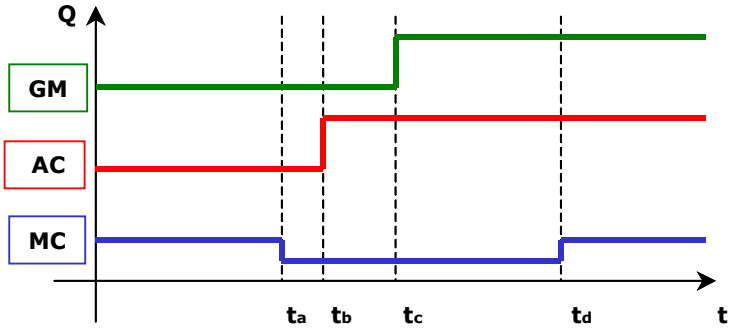

Figure 3. Variations of variables GM, AC and MC over time

5.1.2 Parameter Transmission

Before simulation involving these parameters can be performed according to the DOTS middleware model the parameters must be converted into XML format. The actions specified in the XML files are executed by the adapter on the Optimizer node.

First, an XML schema (XSD) must be defined. XSD contains a data format for the time series is defined. This root element holds: a) An array of elements having the variable names and types from the time series as element names and types, respectively. If exact times at each point in time are required, these can be modeled as DateTime attributes for each element in the array. b) The time step defined as an integer element.

The code excerpt below is an example of how an XSD can be defined for the time series in Figure 3. Line 5 illustrates point a) and line 6 the point b).

```
1.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2.                xmlns:ts="http://mag5.idt.mdh.se/TS"
3.                targetNamespace="http://mag5.idt.mdh.se/TS">
4.
5.   <xs:element name="Parameters" maxOccurs="1">
6.        <xs:sequence>
7.             <xs:element ref="TimeStep" maxOccurs="1"/>
8.             <xs:element ref="TimeSeries"/>
9.        <xs:sequence>
10.  </xs:element>
11.  <xs:simpleType name="TimeStep" type="xs:integer">
12.  </xs:simpleType>
13.  <xs:complexType name="TimeSeries">
14.       <xs:sequence>
15.            <xs:element name="GM" type="xs:integer"/>
```

```
16.           <xs:element name="AC" type="xs:double"/>
17.           <xs:element name="MC" type="xs:double"/>
18.      </xs:sequence>
19.  </xs:complexType>
20.</xs:schema>
```

Once having defined the XSD, different instances if it can be created. An example of an instance is the following:

```
1.  <p:Parameters
2.       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.       xmlns:ts="http://mag5.idt.mdh.se/TS">
4.
5.       <TimeStep>10</TimeStep>
6.       <TimeSeries>
7.            <GM>50</GM>
8.            <AC>10</AC>
9.       <MC>5</MC>
10.      </TimeSeries>
11.      <TimeSeries>
12.      <GM>50</GM>
13.      <AC>10</AC>
14.      <MC>0</MC>
15.      </TimeSeries>
16.</p:Parameters>
```

This instance will be automatically packaged in the `Body` element of a SOAP `Envelope`, which is sent to the Middleware node via Internet. The Envelope also contains the name of the chosen simulator.

5.1.3 Simulation and Output Values

Before the actual simulator on the node can use the variables and their values, they need to be converted from this general format to a format suitable to the simulator at hand. The Middleware Component (MC) on the Middleware node, keeping record of locations and input/output formats for all available simulators, is responsible for the conversion and re-direction of the simulation request. When the simulation has been performed on the chosen simulator, the return values are converted back to the general format, encoded in XML and sent back to the requesting Optimizer node by the adapter on the Simulator node. The adapter on the Optimizer node must transform these results to the optimizer format, before they can be of any use to the optimizer. These actions performed on the output parameters are accomplished according to the same principle as the one for the input parameters.

6 Current State and future work

So far, the design and implementation of MC together with its database have been completed. This implementation also includes an administrative GUI for MC. Moreover, adapters realizing connections between the optimizer and MC as well as between MC and a simulator have been implemented and added to MC. The optimizer and the simulator mentioned here are Tomlab [9] and MATLAB 0, respectively.

The future works include implementation of more connections of the type described above for the rest of the simulators in the DOTS project. In addition, a GUI for adding new simulators would further enhance the system. The final goal is to provide an Integrated Development Environment (IDE) which will facilitate creation of new data types and protocols and re-use already made adapters for particular simulators.

Finally the entire concept needs an extensive verification: Verification of performance, usability and flexibility. In particular a challenge would be to develop u user-friendly development environment which will allow non-programmers to specify and developed complex optimization processes.

7 Conclusion

In order to realize a truly flexible, extensible and interoperable simulation system, a number of challenges such as complex application interfaces, systems running on different platforms and other complexity issues need to be addressed. One way of achieving this is by the distributed model provided by a middleware solution. The case study shows that the model is feasible, with its benefits and limitations.

The model's interoperability features, extensibility and flexibility concerning both simulators and data format are its strengths, while the most concerning issue is its performance; does the distributed model increase or in fact decrease overall performance in the system? The problem applies to small interpolations, which really do not gain much from the model. On the other hand, significant efficiency improvement can be gained if parallel simulations are considered. This issue needs further investigation.

In addition, many of the limitations, such as true platform independence are currently under development and will probably not affect the model in the future. Although the development of the model still is at an early stage and all its features cannot be fully examined, the model seems to be promising.

From the research point of view the project is of interest for software engineering and simulation communities – there is seldom communication between these communities although the results of their research and practice can significantly benefit from synergy of their knowledge.

8 References

[1] *Crnkovic i., Asklund U., Persson A., Implementing and Integrating Product data Management and software Configuration Management Systems, Artech House Publisher, 2003*

[2] Land R., Crnkovic I., Software Systems Integration and Architectural Analysis – A Case Study, IEEE International Conference on Softwre Manintenance, ISCM 2003

[3] Manzoor, K, *A Brief Introduction to .NET,* 2002, http://homepages.com.pk/kashman/dotnet.htm Visited 2002-11-11

[4] Microsoft Corporation, *Overview of the .NET Framework*, 2001, http://msdn.microsoft.com/library/en-us/cpguide/html/cpovrintroductiontonetframeworksdk.asp,Visited 2002-11-11

[5] Skonnard, A., *Understanding SOAP*, DevelopMentor, March 2003

[6] Bordes, W., *SOAP (Simple Object Access Protocol)*, TechMetrix, http://www.techmetrix.com/trendmarkers/publi.php?P=12003, visited 2003-03-26

[7] Jung, F., *XML Backgrounder – technology and applications*, Software AG, 2000, http://www.softwareag.com/xml/about/e-XML_Backgrounder_WP03E0700.pdf, Visited 2003-03-20

[8] *XML Tutorial*, W3 Schools, http://www.w3schools.com/xml/, Visited 2003-03-20

[9] *Tomlab Homepage,* http://www.tomlab.biz/, Visited 2003-05-12

MATLAB Homepage, http://www.mathworks.com/products/matlab/, Visited 2003-05-12