DISTRIBUTED DISCRETE SIMULATION ON THE WEB

Aman Atri[(a)], Felix Breitenecker[(b)], Nicole Nagele[(b)], Shabnam Tauböck[(c)]

[(a)(b)(c)(d)]Vienna University of Technology, Austria

[(a)]aman.atri@tuwien.ac.at, [(b)]felix.breitenecker@tuwien.ac.at, [(c)]nicole.nagele@tuwien.ac.at
[(d)]shabnam.tauboeck@tuwien.ac.at

ABSTRACT
With the vast development of internet technologies web based simulators have come to an extent where flexible loosely coupled components communicate in a distributed way. We are introducing a software architecture where the simulation environment is not limited to browser or applet specific restrictions but using the web as a transport layer. Furthermore this architecture permits us to separate the location of the data collection, the computational part and the visualisation modules. For a more transparent communication between distributed simulators a generic higher level semantics is able to pass information and data on top of a lower-level networking protocol. The idea is to design a system where modules can be deployed and interchanged without redesigning the whole simulator.

Keywords: web based simulation, service oriented simulation architecture, asynchronous communication, dynamic distributed code generation

1. INTRODUCTION

The rising facilities of internet based applications have brought a lot of new paradigms in terms of simulation software. The current trend of simulation frameworks and tools is to get rid of providing only locally restricted simulation environments and to seek for appropriate communication technology for interacting in heterogeneous systems.

Simulation tools can be used within given protocols like HTTP. The current development of web based technologies is leading the World Wide Web to a new era where internet is not only serving the purpose of displaying information in a browser but to interact and exchange information and data. The idea is not to re-invent new communication algorithms but to use existing transfer protocols and to build higher-level architectures which can provide a much more complex and powerful semantics. This paper deals with web based modelling strategies, communication between client and server applications and network solutions for building higher-level simulation environments for larger scalability (Atri 2007, Breitenecker 2006).

The rapid development of the World Wide Web has made it possible that data can now be transferred without constraints concerning target platforms or location of systems.

Distributed simulation on the web requires data exchange from remote components. Thus there will be a detailed view on certain software architectures that are suitable for these scenarios. Complications, problems that do occur in modelling and designing simulations scenarios are also introduced.

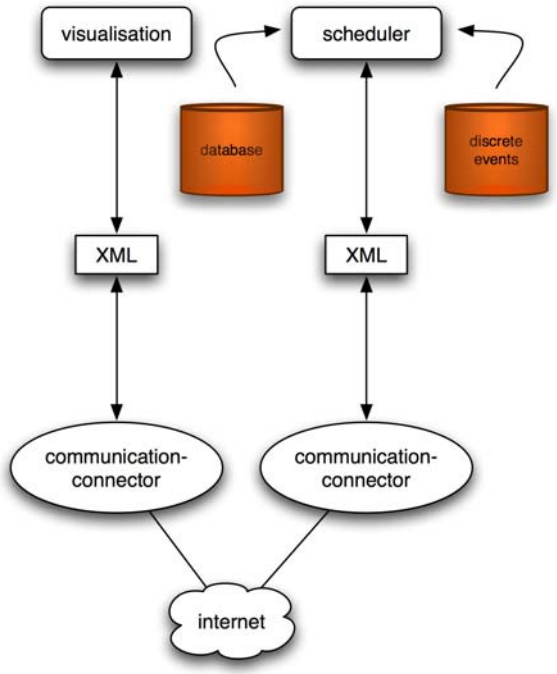

Figure 1: Message passing in transparent networks

2. HETEROGENEOUS COMMUNICATION

Distributed computation demands that our models are spread across the network and they share an internal communication channel to update themselves. Figure 1 shows a loosely coupled framework of a message passing layer which provides an abstraction regarding the dependency towards the actual programming and simulation language. Thus the actual target platform gets independent if we use a globally known and open communication standard like XML (Lechler 99). Now that most of the components do not necessarily share a common physical environment, unified arrangements

must be made so that these components can broadcast their behaviour and presence to other subsystems. Depending on the actual modelling and experiment, web based simulation systems do not only work on broadcasting patterns but can also benefit from subscribing to certain services which are required by the simulation algorithm. This allows the whole system to provide the ability for remote computing within the given context.

2.1. Web based visualization

Due to the behaviour of classical stateless protocols like HTTP data transfer is restricted in direction, chronological order and as a negative side effect it produces a traffic overhead creating potential falsified discrete triggering data. This might result in latency effects in visualization on the client side. There are higher-level approaches of web programming frameworks to implement visualisation trough:

- Dynamical code generation with asynchronous data delivery from the server
- A statically pre-compiled graphical application which can directly access the service and might be able to overtake some portions of the computing unit

The first approach is implemented in Web 2.0 fashion using XML mapping between clients and server. Ajax (Asynchronous JavaScript and XML) emulates the characteristics of a stateless protocol. In case of required data update, the application is able to receive the new visualisation information in background without interrupting the actual simulation process.

The corresponding source code can be generated via the web application on the server side. On the other hand a predefined simulation and visualisation engine is preferred if the data representation is not manipulated from the distributed environment but can be computed on a local system. In the last years this practice has been put using Java Applets embedded in a browser but now it has become more comfortable to develop programs which use native graphical widgets of the operation system and still the application is platform independent. Java Webstart (JWS) simulation applications offer a fast way to synchronize and interact with the simulation service and download the models from the net dynamically (Page and Kreutzer 2005).

The interactive web based simulation enables the user to enter parameters or modify them using the browser. Usually if the application is not running only on the client side the browser sends this parameter to the web server using the HTTP protocol. The request is matched with the unique session of the user and the session values are updated with the latest parameters. The simulation web application is sends the result to the browser and the page is reloaded. Because of this latency during the HTTP requests the visualisation cannot run fluently. Ajax technology is a work around the avoid networking overhead. During the simulation process the data is not changed entirely. The static components which are displayed for documentation purposes remain always the same and only the meta-info graphical part is computed dynamically. For example if the simulation visualisation is represented as a graphical chart where the drawn function is changing its values in certain time intervals and the browser refreshes the page at every new query. All the information which never change like the websites layout style sheet, the cookies etc. will be resent over the network causing a falsified query time stamp because the browser will take its own time to reformat and redisplay the whole page.

If the components of the graphical chart are using Ajax JavaScript then the web application will only receive an XML formatted query where the variables which are necessary for repainting the graph are sent and recomputed without sending the whole HTTP request new. This communication is transported asynchronously and the user has not the responsibility to update and refresh his browser window. The document exists during the entire user session and is only modified within the permitted context. In combination with hyperlinks parts of the visualization can be hidden without reformatting and reading the style sheet commands. The value of the hyperlinks can also be changed dynamically and so they get a semantic quality. Dispatching Ajax queries does not require special browser plugins or extra ordinary configuration. Any modern web browser with enabled JavaScript is capable to deal with asynchronous XML communication. The web application which is computing the simulation results is of course aware of its Ajax capacity. In visualisation where the size of pending data is not known (the simulation time is known but the total amount of computed data might be unpredictable) Ajax helps the simulator to avoid unnecessary traffic overhead.

2.2. Architectures for discrete Simulation

Building simulation networks where unpredictable number of components work together or may drop out implies to design an architecture which implements the following criteria:

- The client side should be simple and contain only a few classes to decrease overhead traffic
- Discrete events triggered remotely have to be recognized and verified in case of data is lost or falsified during data transfer
- Fault tolerance mechanisms have to grant the exchangeability of components in case of loss of connectivity. The simulation engine has to be notified when some remote modules are not reachable

Event oriented systems interact not trough a stream of information and data but with synchronized events and require an architecture which is mostly suitable if we

use the web as a global computation platform. The main focus is not only to optimize single algorithms but the whole system. Event oriented systems don't interact trough a stream of information and data but with synchronized events and require an architecture which is mostly suitable if we use the web as a global computation platform for a frictionless integrity of all storage and computing units (Bass and Clements and , Kazman 1998).

Distributed simulation refers to a system where the components do not only reside on a single system. The communication is done over an internet protocol between the applications. A fault tolerant distributed application is a system where a single component might be out of order but not implicating a shutdown of the whole system. While the retrieval of the faulty component seems to be more or less easy, the actual difficulties rise when we have to look for an appropriate substitution.

2.3. Service-Oriented Simulation

When we talk of web based access to simulation resources we find a lot of end-to-end point implementations where the data is passed over an HTML site of a browser to a CGI script on a server and the computation is done either on the server and sent back to the browser of the client system or we get an inline plugin such as an applet or a flash animation. (Page and Lechler and Classen 2000) The limitation of a duplex client-server architecture is that most of the data might be hidden and we don't have the permission to access them directly. This is were services-oriented architecture comes into practice during a bidirectional message passing when both nodes have to behave not only as a server but also as a client. One implementation of a service-oriented architecture (SOA) is called web services which use an existing networking protocol like HTTP(s). The semantic layer is represented as a XML specification where important information like the names of the classes and methods which are likely to be executed remotely and the parameters and the return values are transferred. In discrete event oriented simulation a sequential concatenation is strategy to implement a distributed waiting queue area (Dustdar and Gall and Hauswirth 2003).

Thus such a simulator works on service-oriented pipes. That means, that every web service offers its functionality to another one. The output of a transaction can then be used as an input for another web service and so on. In case of failure it is very easy to diagnose which chain link is out of order and can be replaced by simply switching to another web service node and redirecting the stream.

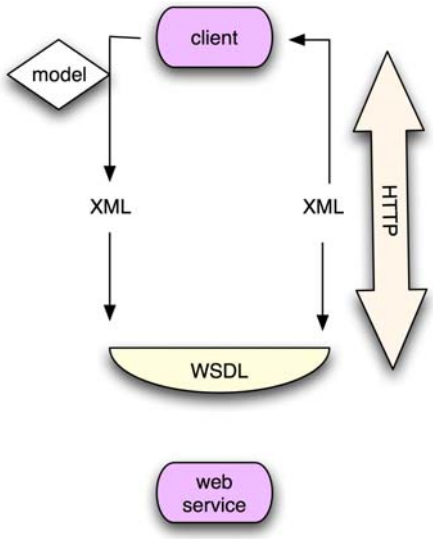

Figure 2: Transaction of a model over SOAP

One big advantage of this concept is the possibility that entities can be handled without depending on a specific programming language. The WSDL specification which provides information about the classes and the interfaces is adopted in most of object oriented languages and frameworks (Gyimesi 2005, Booth et. al. 2004). Especially in discrete simulation we can distinguish the state of a model by its embedded variables and their values. Web services can now provide a manipulation of these variables and programmatic computation without even knowing in which actual language they have been originally configured.

2.4. Sequential and Parallel Piping

Some processes during a discrete simulation computation may require merge and branching of the current model. In that case a web service oriented queue can split the model into several subsets of variables and pass these subsets to different sub-services.

A sequential pipe can branch out to a parallel pipe. That does not imply that the actual calculation is performed as a parallel process but more like a splitted workflow action where the order of the output is irrelevant. If such a service requires some variables from another set, but due to security restrictions the direct access is not granted, then the demanding web service can request a fetch task from the web service which is managing the referring data set. An end-point SOAP station which merges the model together and releases it to the next module is globally accessible.

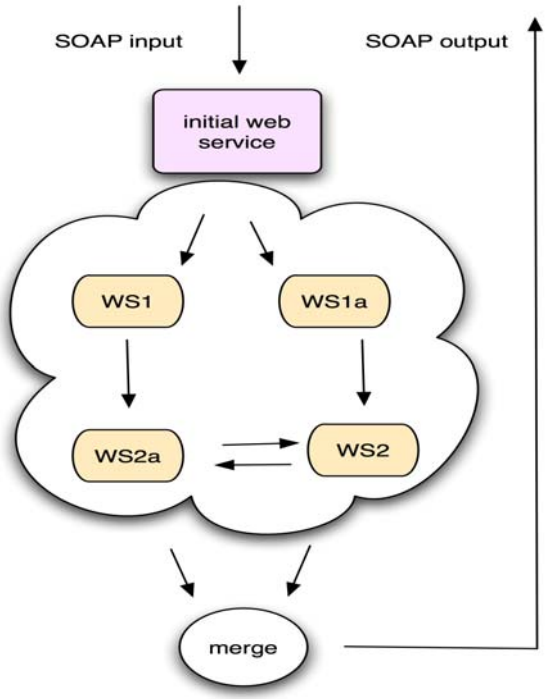

Figure 3: Task divided Simulation Web Services

For any module outside of this constellation the whole system seemed to be a black box. The waiting queue which entered the system had been split and passed on to several different host systems. The policy by which the data is distributed is depending on the current task of the discrete simulation. Such a policy could be driven by access permission of secondary data source which is essential for the computation or the positive side effect of efficient load balancing.

2.5. Multiple persistence of discrete models

In discrete event oriented simulation some tasks are likely to avoid temporal latencies because of quick read and write operations. In many cases the actual state of the model has to be available to all stakeholders in the network. This condition leads to a symbiosis where the following criteria have to be met:

- Every client gets an update of the state of all entities at any time requested.
- Any manipulation of the state of a model has to be transactional. That means every client has to be notified that an update has been committed. If a client does not respond positively than either the transaction has to be withdrawn or the corresponding client has to be listed as non-active.
- Every entity which is accessible to all clients has at least one copy of itself in the system.

2.6. Models in a tupel space

Discrete event oriented simulation works on the communication and message passing between the entities and their sinks and sources. In large scale networks this message passing could get in a bottleneck situation if the whole repository is centralized or the network traffic is unbalanced and some clients might experience latency effects. Furthermore if a centralized repository loses its network connection the whole simulation execution will get stuck. An appropriate mapped out persistence strategy is a so called tupel space as shown in Figure 4.

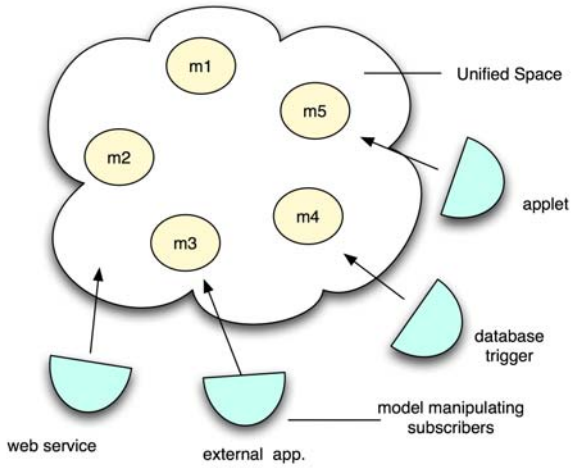

Figure 4: Multiple persistence in a unified space

Somehow in certain work steps the number of read and write operations cannot be predicted. The idea is to unify the memory of all stakeholders to a so called global unified tupel space (TS). Within this space the models are accessible to all memory-providing hosts and are replicated. This means that each model is handled as a unique object in comprehensive higher level memory architecture but actually the tupel space service provides the opportunity that in case of failure of certain parts of the network, the object and data will not be lost and other tupel service nodes will take over the data. Besides that a change of the state of a discrete model (e.g. a modification of the model parameters) will be passed on to every replicated copy in a synchronized fashion.

The programming paradigm of this higher lever architecture was introduced by Dr. David Gelernter at the Yale University as coordination language named Linda. Afterwards a Java based approach was implemented by Sun called Java Spaces. A Java Space service network does not offer a querying language as relational or object oriented database. In discrete simulation the selection of each model is conducted by rules which are based on discrete stochastic events or external temporal depending influences. The main focus of a tupel space is the correct identification of the objects for successful retrieval and transactional synchronisation. The lookup methods for models in

space deliver two types of results: find the exact match or just forget it. A unified memory is quite comfortable for discrete event oriented models using cellular automata (CA) as their state change navigator from the programming point of view because modifications of the current state affecting the whole CA are automatically replicated and updated.

The disadvantage of using Java Spaces is that data transfer within the space itself and with the client is done using Java objects. But building a multi-tier architecture as shown in Figure 4 enables language independent access by inserting a middleware layer which reallocates interfaces like web services or hibernation mapping to external clients. Thus the tupel space can be accessed by any database client, web service client or even directly with an application written in Java like a web based applet or Java Webstart application.

2.7. Publishing and subscription vs. parametric selection

While tupel spaces favour the idea of distributed object storage without interfering with the semantic layer an alternative approach is to allow an external managing software module to allow publishing and subscribing (P/S) to certain objects which represent the state of the model.

Thus we can couple the tupel space storage with external business logic. The concept of P/S is comparable to a subscription system like a newsletter service. The broking system broadcasts its models according to its category. For instance if the discrete simulation is going to compute a distributed waiting queue, a P/S broking system could mange different queues within a total different simulation context. A client could subscribe to a certain queue claiming for notification of only specific types of entities loaded. This can be compared with the waiting queue at an airport. After the passenger enters the airport he will be redirected to the terminal to submit his luggage. This is would resemble our distributed queue. A subscription request is expressed by an airline only for their own passengers. Although the *middleware* (the waiting hall in front of the check-in counters) is managing all passengers of all airlines the counter of a categorized airline can then trigger a notification when a passenger travelling with the corresponding company delivers his luggage.

In object-oriented simulation over a network this pattern is very useful because the P/S broker can handle any arbitrary number of clients. Subscribers gain a profit because they don't have to take care of filtering the search results. They trust their own broker manager who is responsible for the correct selection of the models with the following properties:

- A global queue is storing the events
- Publishers register themselves at the global queue
- Publishers create new events if the state of the model changes
- Subscribers register themselves and describe those events they want to be informed about
- A broker manager who is hired by the subscriber observes the queue and the stored events
- In case the state of a model has been changed due to an event caused by the publisher the broker sends a message to all subscribed clients using either the push or pull method
- *Pull model*: the subscriber receives a notification that data has been changed. The subscription client is now responsible to fetch an update of the model
- *Push model*: the client does not only get the notification but also the whole data bundle.

In event oriented simulation with lots of models states and large objects the pull method reduces the network traffic overhead because the client can filter the content and update only parts of the model which are required for the next simulation process.

2.8. Persistence and Transformation of objects

Time consuming simulation with lot of input and output data require sophisticated persistence of the state of the models. The architectures discussed previously have introduced the exchange of information on different host systems but not the storage of those data models. In terms of large scale discrete simulations in an object oriented programming pattern the state of a discrete model is described by the values of its parameters methods and local and global variables. As a persistent storage platform an object oriented database would be the most appropriate solution. Many systems still do not support object oriented database persistence and store their information in relational databases because:

- Legacy software components would require the whole application to be rewritten
- Performance of relational databases are more efficient in terms of data mining and complex queries

Nevertheless dynamic models where the cardinality of their parameters can increase or decrease during simulation runtime, the object oriented database (OODB) can still recognize and verify these changes, as for the parameter of the model is stored as a complete serialized object. In a relational scheme this is not possible directly because of the static behaviour of the tables and their column definitions.

Hibernation technology plays a great middleware role when old database driven software meets new scalable pure object oriented design pattern. The layer of transforming an object into serialized data (which could be stored on a file system) is interrupted with a transformation of the objects encapsulated data into an

XML declaration of the variables and their visibility properties. Thus the object is transformed into a plaintext readable form where the semantic labels can be reinterpreted. This means that any database driven simulation library can store and read directly from the relational database while an object oriented client can transform (hibernate) the model into an XML form and convert it to SQL statements or into objects. Figure 5 shows the workflow of the hibernation process:

1. The simulator has been provided with a model description.
2. The simulator (or the single component if the simulation is distributed on a network) wants to gain access to data to feed the model.
3. The simulator forwards the model description to the hibernation middleware which has access to a relational database.
4. The hibernation server converts the request into SQL statements and executes them.
5. The result is converted into an object for the simulation client and sent back for further operations.

Thus the information that the data was actually stored in a relational scheme can be hidden and the simulator regards the object locally created instance.

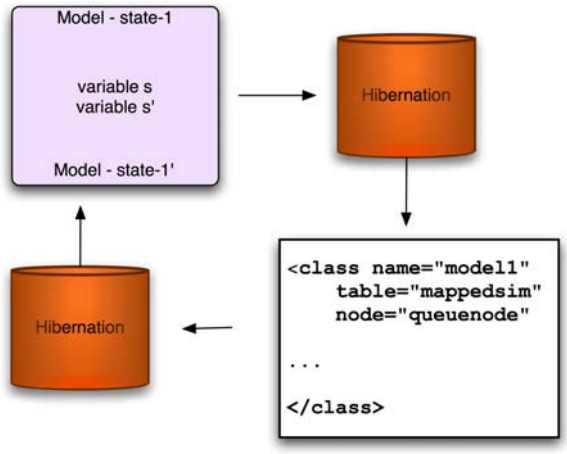

Figure 5: Dynamic mapping of model entities

3. CONCLUSION

We have shown that a composition of different architectural designs can build a highly scalable network for distributed simulation. Software architectures for discrete simulation are not merely focusing on parallel computation but also asynchronous exchange of the models and state modification. Higher level architectures can hide the actual machine representation of the model and operate only on the semantic value of the objects. Network transparent protocols can hide the complexity of the data transfer. For the simulator the actual location of the object is transparent and the models are treated as they would reside on the local system and the local runtime.

REFERENCES

Page, B., Kreutzer, W., 2005. *The Java Simulation Handbook. Simulating Discrete Event Systems with UML and Java*. Aachen:Shaker.

Page, B., Lechler T., Claassen S., 2000. *Objektorientierte Simulation in Java mit dem Framework DESMO-J Java.* Norderstedt:Books on Demand GmbH.

ARGE Simulation News, *ARGESIM Comparisons on Simulation Technique and Tools.* TU-Wien. Available from: http://www.argesim.org/comparisons [accessed 27 April 2008].

Atri, A., 2007. *Visualisierung verteilter diskreter Simulationen im Web*. Thesis (master). Vienna University of Technology.

Breitenecker, F., 2006. *Software for Modelling and Simulation - History, Developments Trends and Challenges.* Conference on Simulation and Visualization 2006, pp. 7-20. March 2-3 Magdeburg (Magdeburg, Germany).

Gyimesi, M., 2005. *Simulation Service Providing unter Verwendung von Web Service Technologie.* Thesis (PhD). Technische Universität Wien.

Booth D., Haas, H., McCabe, F., Newcomer E., Champion, I.M., Ferris, C., Orchard, D., 2004. *Web services architecture.* Technical report W3C – World Wide Web Consortium. Available from: http://www.w3.org/TR/ws-arch/ [accessed 27 April 2008]

Dustdar, S., Gall, H., Hauswirth M., 2003. *Software-Architekturen für Verteilte Systeme.* Berlin: Springer.

Lechler T., 1999. *Entwurf und Implementierung eines Frameworks für diskrete Simulatoren in Java*. Thesis (master), Universität Hamburg.

Bass L., Clements P., Kazman R., 1998. *Software Architecture in Practice*. Boston:Addison Wesley.

AUTHORS BIOGRAPHY

Aman Atri studied Software and Information Engineering at the Vienna University of Technology.
His bachelor thesis analyses automated proofs for model checking in temporal logic. After his bachelors program he pursued with the master course Software Engineering and Internet Computing. His master thesis deals with discrete simulation and visualisation schemes for the web. This work has been dilated in his PhD thesis where he is analysing and developing service oriented simulation frameworks and interoperable connectivity of different simulators. He is working as an assistant at the Vienna University of Technology (Institute for Analysis and Scientific Computing).