Visualizing and Interacting with Multiple Simulations using the Multichronic Tree

Abstract - It has been proved experimentally that Visual interactive simulation (VIS) is useful in specific contexts. Interaction with the simulated system remains one of the key issues that need to be addressed for VIS to be applied on a large scale. Little work has been reported recently on using modern computers that offer new types of user interfaces for solving the problem of interactivity with simulations. This paper introduces the multichronic tree user interface concept that provides users with tools for interacting with and visualizing multiple simultaneously running simulations. This new VIS concept builds on previous systems, formalizes operations and adds several interactive features that can be supported by state-of-the-art computer architectures and display environments. A prototype of the multichronic tree concept is presented along with a discussion on how it could be integrated with other tools in a more general framework for helping users in their simulation exploration process.

Introduction

Visual interactive simulation (VIS), in opposition to formal statistical analysis of simulation experiments, is the concept of providing users with the opportunity to change intuitively simulation model parameters and observe the results of these changes directly (Bell and O'Keefe, 1995). The use of VIS in a simulation experiment is a controversial issue since it is claimed that users' cognitive processes may be biased during the experiment, which may lead to wrong conclusions on the experiment results or to overestimation of users' competence (Bell and O'Keefe, 1995, Dörner, 1996).

When supported by appropriate tools and human-computer interfaces (HCI), though, VIS has the potential to help users understand their system's models. Moreover, interacting dynamically with running complex simulation models favors users' comprehension of the model better than when they are not included in the experiment loop (Darabi et al., 2007). Including users in the experiment loop implies that they are requested to participate interactively in the simulation process via a rich and intuitive HCI. Key principles of human-centered design such as simplicity, interactivity and affordances must therefore be thoroughly implemented in the VIS tool to make it both usable and useful (Norman and Draper, 1986). An intuitive tool for exploring simulations should also

exploit the ability of the human perceptual system to recognize patterns in graphical representations of information (Dane and Pratt, 2007).

The work presented in this paper aims at developing a collection of tools for exploring simulations of complex systems or phenomena. The use of these tools, akin to the use of formal simulations experiments tools, should help users to develop a basic understanding of a simulation model, elaborate robust decisions or policies, and compare the merit of the outcomes of various decisions (Kleijnen et al., 2005). The level of understanding may range from scoping a broad problem to identifying singular behaviors of specific model components (or parameters). Through this multi-level exploration process, users can deepen their perception of a phenomenon through parameter space and simulation space exploration. It is worth mentioning that the approach described in the next sections does not replace formal simulation experiments. It rather provides a rich collection of tools for users to explore better the systems under study and their underlying complexity. In a broader context, the proposed approach could be the first exploration step in a data farming process (Horne and Meyer, 2004).

In the early days of VIS - the early 90s - limited computer performance hindered the implementation of informative and responsive software modules required by VIS tools as suggested by Kalawsky et al. (Kalawsky and Nee, 2004). However, user interfaces are easier to develop today with increased computational power and sophisticated frameworks. With these principles in mind, a set of features that should be offered to users by a VIS tool can be defined. The VIS user interface described in this paper, called the multichronic tree, takes its roots in the history tree concept proposed in GRASPARC (Brodlie et al., 1993). However, it departs from the history tree in both functionality and implementation and adopts a user-centered instead of a simulation-centered design.

Our work focuses on simulation scenarios that execute in seconds or minutes, in which the "time" dimension is important. In opposition for formal computer experiments that do not cope with decisions made at runtime due to exponential increase in the number of modifiable parameters, our system offers a wide range of tools aimed at making and analysing decisions during a course of action. The VIS tool proposed in our work not only aims at measuring effectiveness at the end of a simulation run but for the entire course of the simulation during which users' decisions can be taken into account and may result in the occurrence of unpredictable events. Also, we consider simulation models comprising characteristics of complex systems such as nonlinearity and

strong coupling between elements (Batty and Torrens, 2005), for which formal simulation experiments usually fail. Agent-based distillations are particularly well-suited for such applications (Sanchez and Lucas, 2002, Horne and Meyer, 2004).

The overall objective of the current work is to define a formal representation relevant to the new multichronic tree concept, and to develop and implement functionalities on the tree that can be useful to users in reaching successful simulation experiments. Section 0 presents previous work on VIS. Section 0 presents an overview of the multichronic tree concept as well as the features that it should implement. Elements that should be considered in choosing a simulator able to support the multichronic tree are also discussed. A prototype implementation is presented in Section 0. Future work and challenges that need to be addressed in refining the multichronic tree concept are discussed in Section 0.

Previous Work

The literature on user interfaces for visual interactive simulation is rather scarce. To our knowledge, GRASPARC was the first project that has proposed a user interface using a history tree for representing multiple checkpointed simulations (Brodlie et al., 1993). Checkpointing is a fundamental concept introduced by GRASPARC which consists of saving the current status of a simulation in a file, in a database, or in memory, for later use. Checkpointing is of paramount importance since it supports the user in the discovery process in an iterative simulation. With checkpointing, the user is entitled to making wrong decisions during his interaction with the simulation and to go back in time for choosing a new set of values for simulation parameters without invalidating the entire simulation exploration process. The history tree, as a visual representation of the available checkpoints, solves the problem of tracking the parameter space exploration. GRASPARC aimed at building a problem-solving environment having the history tree as its main user interface. However, the use that is made of the history tree is not the most intuitive and the tree does not seem to integrate smoothly with other components of the system. Moreover, the authors do not provide much detail on how the user interface can be exploited and on the operations that are supported by the interface.

Wright *et al.* introduce HyperScribe, which integrates GRASPARC's history tree concept into the IRIS Explorer software suite (Wright and Walton, 1996). The authors claim that this integration with an existing

application makes the history tree much simpler to use. A major problem with HyperScribe is that the tree is no longer the central interaction point with the simulation, but is rather exploited as any other tool available for managing the data pipeline in the IRIS Explorer suite. This makes using the tree as a tool for navigating and exploring a simulation difficult.

As demonstrated in the next sections, we claim that a wide range of operations can be performed by users on the history tree as the main interaction point with computational components of a simulation. Dangelmaier *et al.* (Dangelmaier et al., 2006) exploit the concept of simulation cloning which consists of launching the execution of several simulations with customized input parameters in parallel. Simulation clones are represented in a "configuration tree", which shares similarities with the history tree in GRASPARC. The configuration tree allows users to start/stop a configuration, create new branches and prune instances of simulations. Since this work is in progress, there are no additional details on other available functionalities.

Kalawsky *et al.* (Kalawsky and Nee, 2004) discuss important issues that must be addressed by interactive user interfaces in computational systems. They report that a user interface should ideally respond within 0.1 s so that the user does not feel any delay in the response he expects from the system. A delay of 1 s is the maximum acceptable delay to avoid interrupting (and disturbing) the user's line of thought. These performance targets were considered in the design of the multichronic tree user interface.

The work proposed in this paper defines and formalizes a set of high-level operations that can be supported by a tree-like representation of both the parameter space and the simulation space in a simulation exploration process. Since this new concept allows for multiple courses of actions to be run and observed concurrently, the tree-like representation is called a multichronic tree (Multichronic comes from Greek roots "multi" for "many" and "chronic" for "time"). The multichronic tree acts as an interface for monitoring concurrent simulations. It provide users with means of highlighting the relationships between simulation instances as well as tools for exploring the data generated by these instances while they execute.

The multichronic tree user interface

Figure 1 shows a generic multichronic tree with its more important elements along with their definition. Although it is inspired from the

history tree proposed in GRASPARC (Brodlie et al., 1993), the multichronic tree representation can support several additional interactive features that are described in the following sections.

Formal representation

As depicted in Figure 1, a tree originates from a root node (A) – i.e. the root simulation under study – which can own children (C) resulting from changes brought to parameters in the root simulation. Changing the value of a parameter can result in the creation of a divergence point (I and J) for which new simulations are spawned from a running simulation branch. A parameter can also be changed without spawning a new simulation branch (H). It is also possible to request that the value of a parameter change continuously (O). Each leaf of the multichronic tree is associated with a simulation on a row (D), while every node is linked by a branch to its parent (B), each node is associated with a specific set of simulation parameters.

A reference axis (G) is used as a reference on which simulations can be aligned with each other to ease the comparison and exploration process in the multichronic tree. Time is the most common parameter that is used as the reference axis. A simulation evolves along this axis (and has its time parameter set to value (N) for instance). This axis can be used to compare multiple simulation instances, either directly (F) or through a window of values (E) of the reference parameter. Finally, every node owns an ID string describing the change that has been brought to a parameter. For clarity, these ID strings are omitted in Figure 1.

Additional elements related to the multichronic tree concept, which are not shown in Figure 1, are defined as follows:

- Variable: any data element that can be modified by either the user or the simulation;
- Parameter (input variable): any simulation variable that can be modified by the user;
- Metric (output variable): any variable, or combination of variables, that has been modified by the simulator during the execution of a simulation;
- Variable of reference: variable associated with the axis of reference (G).

Some of the above elements have been defined differently by other authors. For instance, a "divergence point" in the multichronic tree is usually referred to as a "decision point" in other systems. We claim that the word *Divergence* is more relevant since it refers to the point in the simulation where two concurrent simulations diverge in terms of simulation metrics resulting from calculations, a result that is not

necessarily generated by an intervention of the user but rather by an automated process during the experiment. A constraint that is imposed on the multichronic tree is that the value of the variable of reference should increase/decrease monotonically (as time does in a simulation). The next sections describe the operations that can be supported by the multichronic tree in the process of exploring simulations interactively. Implementation issues are also discussed.

Operations supported by the multichronic tree

The multichronic tree concept supports the interactive exploration of simulations through the implementation of a set of operations. These operations are described in the following enumeration based on the formal representation defined in the preceding section. It is important to insist on the fact that the purpose of the multichronic tree is not only to allow users to explore the parameter space of a model or system that is simulated but also to offer ways of exploring the *simulation space* which consists of all the courses of actions being generated by concurrent simulations in a study and their associated data.

- *Create/Destroy a divergence point*: at a given moment in time, a user may decide to modify a simulation parameter in order to perform a "what-if" inquiry or to take a decision regarding an event happening in the course of action. According to Figure 1, this action *creates* a divergence point in the tree. The initial simulation continues unmodified while a cloned instance runs with the new value of the parameter. Similarly, a user may decide to discard a subset of a course of actions by *destroying* a divergence point.
- *Change a simulation parameter*: at a given moment in time, a user may decide to modify a simulation parameter without creating a new branch. This operation should be used when the original course of action is not of interest anymore for the user.
- *Vary a parameter continuously*: a user may want to observe the effect of changing a simulation parameter in a continuous way. This operation is similar to computational steering (van Liere et al., 1996).
- *Duplicate a course of actions*: a user may want to duplicate parameter changes performed onto a given course of actions to another one. Users should however be cautious with this operation since this could lead to temporal inconsistencies in the replicated course of action (due, for instance, to hidden coupling between parameters).
- *Support visual playback*: the user is interested in viewing what happens to the data in a simulation. This data can be displayed as a snapshot of a given simulation time step or be more complex in terms of representation (e.g. multidimensional metaphor).

- *Simulate and playback several courses of actions simultaneously*: a user may want to start *multiple* simulations at the same time by varying parameters and playback the resulting data asynchronously.
- *Pause/Resume/Stop a simulation playback*: a user may decide to pause the simulation playback and then have it resumed. Also, when the user considers a course of action as being completed, he can stop the simulation playback, leading to a resulting course of action.
- *Go forward and backward in time*: this feature allows users performing a step by step analysis of what is happening in a simulation by jumping from one simulation time to another instantaneously. Rewinding a simulation playback allows examining a past course of actions. Operations such as creating a divergence point are thus possible even though an event occurred in the past and was considered as being "lost".
- *Vary the simulation playback speed*: this feature allows focusing on specific intervals of a simulation while skipping other segments rapidly.
- *Change the playback level of details*: a user may be interested in analysing only high-level events in a simulation or, on the opposite, focusing on very low-level details.
- *Change the variable of reference*: a user may want to switch between different variables of reference thus allowing simulation playbacks to be compared over a custom metric or parameter other than time.
- *Synchronize multiple simulation playbacks*: a user may want to compare events occurring in two or more simulation playbacks. In order to achieve this task, the playbacks need to be synchronized with respect to the reference variable.
- *Change a simulation context*: since a problem can include several components located at different levels of simulation, several types of multichronic trees can be instantiated and can communicate together in order to exchange useful meta-information.
- *Collapse/Expand sections of a tree*: when a tree becomes cluttered, a user may want to hide parts of it by making branches collapse at selected nodes.
- *Instantiate additional multichronic trees in different work sheets*: additional work sheets can be added to a workspace, thus allowing a user to segment his exploration over a problem.
- *Change the layout of the tree*: the layout, which organizes nodes and branches vertically and horizontally on the display screen, can be customized in order to view different perspectives of a work session.
- *Assign relevant labels to nodes and branches*: since a multichronic tree can contain a large amount of nodes, being able to identify the operation associated to every one with a relevant label is essential.
- *Bookmark/Take a note over elements in the tree*: a user may want to take a note during the simulation experiment.

- *Save/Load a work session*: after several hours of work, a user may want to save his session in order to return to his work at a later time or to share it with other users.
- *Keep a log of user actions*: a user interface like a multichronic tree can be used in experiments aiming to study how an individual addresses a simulation. Playing back user actions would be of great interest for researchers responsible for the analysis of the actions performed by the users participating in the experiments.

The above list provides an exhaustive overview of the operations a user would be able to perform on a full-blown implementation of the multichronic tree interface. However, specific simulation engines or the context in which the simulation experiments are conducted may impose constraints that could hinder the implementation of some of these operations. The next section describes the requirements that simulation engines should satisfy in order to allow for the implementation of the operations supported by the multichronic tree representation.

Constraints on simulation engines

The previous section presented operations that the multichronic tree user interface can implement. Depending on the simulation engine used and the context of a simulation experiment, most operations are feasible and relevant. However, the simulator with which the multichronic tree interacts for supporting these operations, and from which data is extracted, should implement several features to allow users to benefit from the multichronic tree representation. Table 1 lists constraints that should be satisfied by a simulator that interacts with the multichronic tree user interface.

The most important constraint is relevant to the creation of divergence points. If the simulator does not support this feature, it is impossible to generate new simulation data interactively. It should be noted that if the simulator does not allow for the creation of divergence points at runtime, the multichronic tree user interface can always be used as a playback tool offering operations such as moving forward/backward and synchronizing several simulation playbacks in time.

Making user interaction with the multichronic tree easier through automated rules

Automated rules should be used as support during the parameter/simulation space explorations in order to relieve users from performing repetitive or tedious tasks on the multichronic tree. For

example, a user might want systematically to create a new divergence point once a given simulation metric reaches a certain threshold. Building a rule accomplishing this task thus relieves the user from monitoring the value of the metric.

A rule is composed of three parts: a condition, a conclusion and an action. Depending on the state of the multichronic tree, the operations performed by the user and the data generated by the simulator instances, a condition evaluates to "true" or "false". On a "true" evaluation, a conclusion is triggered, which leads to the execution of one or more actions. Examples of actions are the creation of a divergence point, the pause of all simulation playbacks, the pruning of a course of actions, etc.

Rules can be of different types:

- *Automatic and global*: the rule can be applied to every running simulation;
- *Automatic and local*: the rule can be applied to a specific set of running simulations;
- *Manual*: the rule is triggered by user interaction.

Automatic rules are typically associated with conditions triggered by data generated by simulations, whereas manual rules are associated with conditions that are "always true" or conditions that depend on other user interaction events.

For each rule activation, an associated event is added to a log that stores a record of the work accomplished by a user on the multichronic tree interface. Also, more sophisticated algorithms can be envisioned for automatic parameter space exploration of complex simulations. Genetic algorithms are potential candidates for solving such problems (Szczerbicka et al., 1998).

Multichronic tree implementation

A prototype implementing most of the multichronic tree operations has been developed and integrated into an application called Multichronia (Rioux et al., 2008b). It exploits the Java-based Processing library (Processing, 2008) for displaying the multichronic tree (nodes, branches, labels and axis of reference) and for supporting user interaction with the tree for executing operations (create new divergence points, control simulation playback, etc.). The Phylowidget library (Phylowidget, 2008) is used to implement interactive controls such as menus and toolbars. Figure

2 shows interactive controls and a circular menu that are used to interact with the multichronic tree and control the simulations playback.

The current prototype implementing multichronic tree concepts and operations exploits the Pythagoras simulator (Pythagoras, 2006), which is a multi agent-based distillation engine generally employed for simulating military scenarios. Many reasons have motivated this choice. Firstly, it is an open source tool which makes interfacing with the multichronic tree easier. Secondly, in Pythagoras, an XML file with a predefined schema is used for describing a simulation scenario. Hence, reading and changing initial conditions of the simulation is straightforward. Thirdly, Pythagoras already includes a simulation playback viewer that shows agents' position and health, as well as the terrain on which they evolve.

Pythagoras was modified in such a way that it could be interfaced with the multichronic tree. For instance, the functionalities of saving a simulation while it is running and of resuming the simulation have been added to the simulator. Another enhancement that has been brought to Pythagoras is the ability to log and replay a simulation on demand. These first two modifications exploit XML technologies such as Java XML binding (JAXB, 2008). Extra elements were added to the XML schema for saving an executing simulation state. Appropriate methods have also been implemented for enabling simulation checkpointing. The final important feature that was added to Pythagoras is the capacity to communicate with the software implementation of the multichronic tree. This is achieved by a network socket that connects to a server which accepts command messages, and replies by sending appropriate data (e.g. simulation metrics, message replies). Command messages are predefined and are generic enough so that they can be adapted to other simulation packages than Pythagoras.

Figure 2 shows the look of the current implementation of the multichronic tree concept. Every simulation executes with a specific set of parameters and is identified by a label, which is the name of the parameters that have been modified along with their values. The label's background color provides information on the current status of a simulation playback. A red background indicates that a simulation playback is paused, a green one indicates that it is running, and a black one indicates that it is stopped. Further information on the scope of an operation can be found in the label's stroke, such as which node is currently selected. Right-clicking in the multichronic tree opens a circular contextual menu (see Figure 2), which displays available operations in the current context.

Other interaction features such as saving and loading a workspace, changing the tree layout and other configuration options are accessible via menus as shown in the screenshot. Additionally, a scrolling number field adjusts the simulation playback speed. The axes of reference are drawn at the bottom of the screen. In Figure 2, "Simulation time" constitutes the reference variable with which the layout algorithm horizontally aligns the nodes. On the other hand, the "deadBlues" axis shows information about a specific metric. Moreover, the synchronization link indicates that the two corresponding simulations always evolve together with respect to simulation time. Other tools for scrolling and zooming are available to users in order to detail parts of the tree. Finally, the architecture of the system, which relies heavily on design patterns, allows easy implementation of additional operations/features.

Multichronia is currently in the process of being revamped so that the use will be more intuitive and users more effective in their exploration of a given problem. Notably, additional feedback mechanisms have been implemented, providing users with additional clues on the status of their simulations.

The Multichronia software architecture includes a configurable data pipeline based on an XML data representation (Rioux et al., 2008a). This data pipeline consists of several data transformation units, each accomplishing a specific functionality. It also includes a conceptual data model for facilitating the development of exploited simulators. The Pythagoras simulator is actually exploited by the data pipeline, which allows for data to be archived in a SQL database in order to ensure data persistence and retrieval of data for a posterior analysis. Moreover, the pipeline is generic so that additional simulators could interface with the multichronic tree without the need for reengineering Multichronia.

Future work and challenges

The main objective of the work presented in this paper is to provide users with a set of tools in an integrated workspace for exploring simulations of complex systems in an intuitive and interactive manner (Bar-Yam, 2003). We claim that the implementation of the operations supported by the multichronic tree form solid foundations for reaching this goal. The implementation of a first prototype of Multichronia demonstrates the basic concepts of the Multichronia approach.

A major key element from which Multichronia will benefit when completed is the capacity to display the outcome of multiple simulations in the same visualization environment. This module, supported by other elements of the Multichronia workspace, should provide users with rich visualization tools that could improve their understanding of the system that is being simulated. This module is currently under development and allows displaying an improved view of a given course of actions, when compared with the Pythagoras viewer. It is planned to implement the visualization environment on 4-wall immersive virtual reality room that can be transformed on a 32 feet wide wall.

Finally, it is planned to conduct experiment with human to assess the gain in user performance that can be achieved by the Multichronia environment with immersive visualization interface compared to other methods (data farming, computational steering, etc.). With the help of cognitive psychologists, we will be able to determine which of the proposed tools and functionalities are the most helpful and thus optimize the cognitive load of users when exploring a complex problem.

References

BAR-YAM, Y. (2003) Unifying Principles in Complex Systems. *Converging Technology (NBIC) for Improving Human Performance, MC Roco and WS Bainbridge, Dds., Kluwer.*

BATTY, M. & TORRENS, P. M. (2005) Modelling and prediction in a complex world. *Futures,* 37, 745-766.

BELL, P. C. & O'KEEFE, R. M. (1995) An Experimental Investigation into the Efficacy of Visual Interactive Simulation. *Management Science,* 41, 1018-1038.

BRODLIE, K., POON, A., WRIGHT, H., BRANKIN, L., BANECKI, G. & GAY, A. (1993) GRASPARC-A problem solving environment integrating computation and visualization. *IEEE Visualization*, 102-109.

DANE, E. & PRATT, M. G. (2007) Exploring Intuition and its Role in Managerial Decision Making. *The Academy of Management Review (AMR),* 32, 33-54.

DANGELMAIER, W., HUBER, D., LAROQUE, C., AUFENANGER, M., FISCHER, M., KROKOWSKI, J. & KORTENJAN, M. (2006) d3FACT insight goes parallel Aggregation of multiple simulations. *SimVis.*

DARABI, A. A., NELSON, D. W. & PALANKI, S. (2007) Acquisition of troubleshooting skills in a computer simulation: Worked example

vs. conventional problem solving instructional strategies. *Computers in Human Behavior,* 23, 1809-1819.

DÖRNER, D. (1996) *The Logic of Failure: Recognizing and Avoiding Problems in Complex Situations*, Perseus Press.

HORNE, G. E. & MEYER, T. E. (2004) Data farming: discovering surprise. *Winter Simulation Conference.*

JAXB (2008) Java Architecture for XML Binding (JAXB).

KALAWSKY, R. S. & NEE, S. P. (2004) Important issues concerning interactive user interfaces in grid based computational steering systems. *Proceedings of the UK e-Science All Hands Meeting.*

KLEIJNEN, J. P. C., SANCHEZ, S. M., LUCAS, T. W. & CIOPPA, T. M. (2005) A User's Guide to the Brave New World of Designing Simulation Experiments. *INFORMS Journal on Computing,* 17, 263-289.

NORMAN, D. A. & DRAPER, S. W. (1986) *User centered system design*, Lawrence Erlbaum Associates Hillsdale, NJ.

PHYLOWIDGET (2008) PhyloWidget: Web Visualization for the Tree of Life.

PROCESSING (2008) Processing 1.0 (BETA).

PYTHAGORAS (2006). http://www.projectalbert.org.

RIOUX, F., BERNIER, F. & LAURENDEAU, D. (2008a) Design and Implementation of an XML-Based, Technology-Unified Data Pipeline for Interactive Simulation. *Winter Simulation Conference.* Miami, FL.

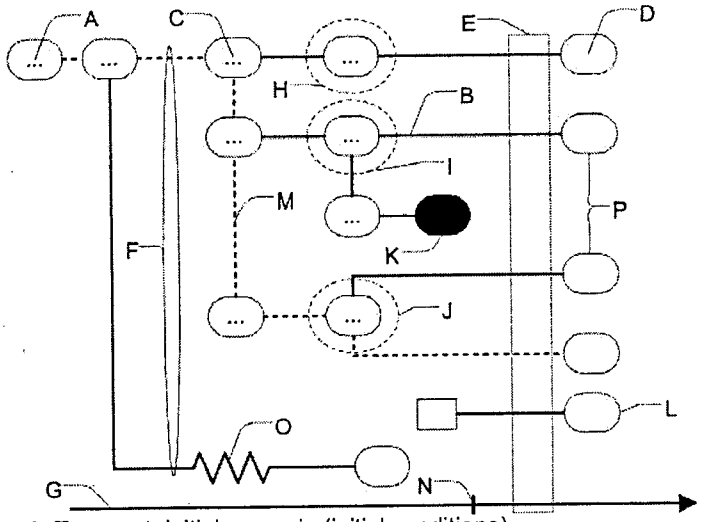
RIOUX, F., BERNIER, F. & LAURENDEAU, D. (2008b) Multichronia - a Generic Parameter, Simulation, Data, and Visual Space Exploration Framework. *Interservice/Industry Training, Simulation & Education Conference.* Orlando.

SANCHEZ, S. M. & LUCAS, T. W. (2002) Exploring the world of agent-based simulations: simple models, complex analyses. *Simulation Conference, 2002. Proceedings of the Winter,* 1.

SZCZERBICKA, H., SYRJAKOW, M. & BECKER, M. (1998) Genetic Algorithms: A Tool for Modelling, Simulation, and Optimization of Complex Systems. *Cybernetics & Systems,* 29, 639-659.

VAN LIERE, R., MULDER, J. D. & VAN WIJK, J. J. (1996) *Computational Steering*, Computer Science, Department of Interactive Systems, CWI.

WRIGHT, H. & WALTON, J. (1996) HyperScribe: A data management facility for the dataflow visualization. IN IETR/4, I. E. T. R. (Ed.). IRIS Explorer Technical Report IETR/4, NAG Ltd.



A. Tree root: initial scenario (initial conditions)
B. Tree branch: partial course of action
C. Tree node: divergence point
D. Tree leaf: simulation instance
E. Parallel segments: window of comparison
F. Parallel branches: aligned points
G. Horizontal axis: axis of reference
H. Action of modifying a simulation parameter: simple direct modification
I. Action of creating a new divergence point: divergence point creation
J. Action of multiple simulation parameters modifications: multiple direct modification
K. Terminated simulation: resulting course of action
L. Pruned simulation: insignificant course of action
M. Path from the root to the leaf: course of action
N. Location on the axis: point
O. Zigzag sign: continuous parameter change
P. Moving nodes: simulation playbacks

Figure 1 – Generic multichronic tree

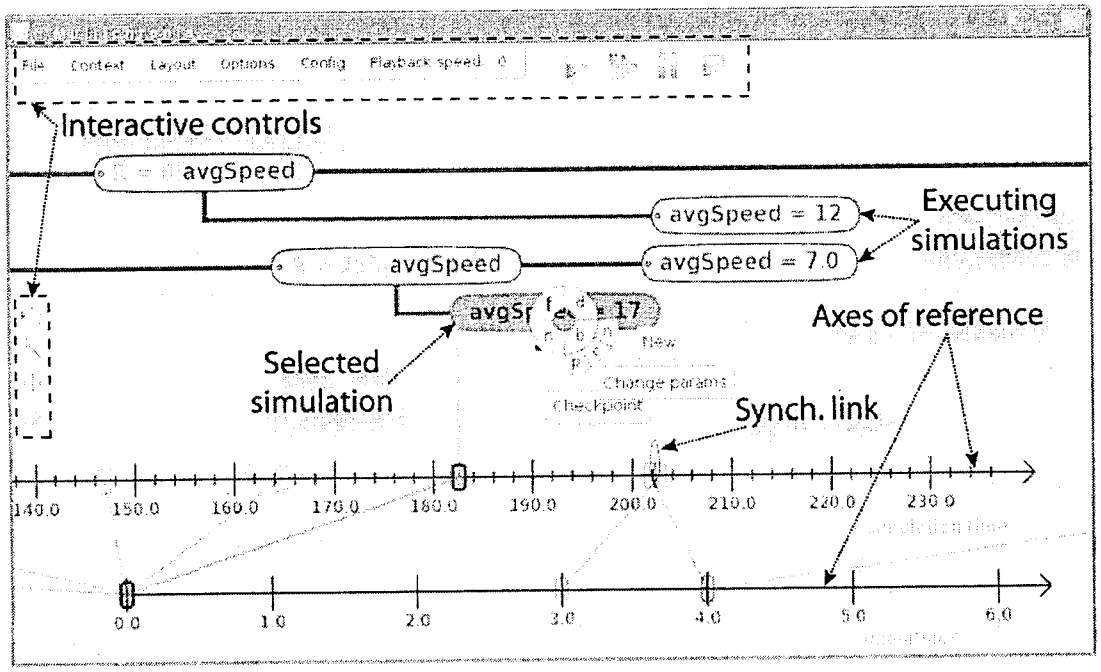


Figure 2 - Screenshot of Multichronia during a simple parameter space exploration

Table 1 - Constraints imposed to a simulator in order to implement the multichronic tree user interface

Multichronic tree operation	*Constraint on the simulator*	*Note*
Create a divergence point[1] and vary a parameter continuously	Allow for direct parameter change and simulation cloning	A simulator would allow for 1) cloning the original simulation and 2) direct parameter change so that the new course of actions would be modified directly
Create a divergence point[1]	Allow for checkpointing and reloading data (optionally at any given simulation time)	A simulator should allow for 1) checkpointing (the process of dumping a simulation's execution to a file) 2) modifying the resulting file and 3) reloading the modified version to another instance of the simulator
Simulation playback	Send simulation data to external software modules	A simulator should send simulation data asynchronously to the multichronic tree, so the user is able to play back the simulation
	Playback environment available	In order for users to observe what is happening in a simulation, a playback environment is essential. Otherwise, a custom visualization environment should be developed
Simultaneous simulation of several courses of actions	Multiple instances running on the same computer	When using multiple CPUs, it should be possible to run multiple simulation instances on one computer
Generic	Open source code	When source code is open, it is easier for developers to instrument the simulator

[1]Either one or the other of the two constraints should be satisfied for implementing this operation