Definition of Real-Time Simulation in the CD++ Toolkit

Ezequiel Glinsky

Departamento de Computación
FCEN – Universidad de Buenos Aires
Planta Baja. Pabellón I.
Ciudad Universitaria (1428)
Buenos Aires. ARGENTINA.
eglinsky@dc.uba.ar

Gabriel Wainer

Dept. of Systems and Computer Engineering
Carleton University
4456 Mackenzie Building
1125 Colonel By Drive
Ottawa, ON. K1S 5B6. CANADA.
gwainer@sce.carleton.ca

Keywords: Discrete event simulation, DEVS, real-time, flattened simulation.

Abstract

The CD++ toolkit was developed in order to implement the theoretical concepts specified by the DEVS formalism. The existing simulation technique available in CD++ employs a virtual time approach. This work presents the definition and implementation of real-time simulation to the toolkit and ties the model execution to a wall-clock attached to the system (physical time). The new simulation technique allows the interaction between the model and its surrounding environment. Time constraints can be easily imposed by the user to the simulated system. A non-hierarchical simulation approach is also presented and introduced to CD++ to reduce the communication overhead. The new flattened simulation technique allows a more effective execution when the real-time approach is used. These recent enhancements are detailed and tested here.

INTRODUCTION

The **DEVS** (Discrete EVents Systems specifications) formalism [Ziegler et al., 2000] provides a framework for the construction of hierarchical models in a modular fashion, allowing model reuse and reducing development and testing time. DEVS models can be executed using abstract simulation mechanisms independent of the model itself. Models are built using a set of basic models called **atomic**, which can be combined to form **coupled** ones. A DEVS atomic model is described as:

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, D >$$

Here, ***X*** is the input events set, **S** is the state set, and ***Y*** is the output events set. There are also several functions: $\boldsymbol{\delta_{int}}$ manages internal transitions, $\boldsymbol{\delta_{ext}}$ external transitions, λ the outputs, and **D** the elapsed time.

A DEVS coupled model is defined as:

$$CM = < I, X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\} >$$

Here, ***X*** is the set of input events, and ***Y*** is the set of output events. **D** is an index of components, and for each i $\in$ D, $\mathbf{M_i}$ is a basic DEVS model, where $M_i = < I_i, X_i, S_i, Y_i, \delta_{int i}, \delta_{ext i}, ta_i >$. $\mathbf{I_i}$ is the set of influencees of model i. For each j $\in$ I_i, $\mathbf{Z_{ij}}$ is the i to j translation function.

The **CD++** toolkit [Rodriguez and Wainer 1999, Wainer et al., 2001] implements DEVS theory. A specification language allows the creation of coupled models, the initial configuration for the atomic models, and the creation of external events to be used during the simulation. Lately the CD++ tool has been enhanced to support parallel simulation [Troccoli and Wainer 2001] and this work presents the real-time enhancements.

DEVS provides the advantages of a discrete event approach in terms of execution performance. Discrete event models evolve in continuous time. Events are instantaneous and can occur asynchronously at unpredictable times. DEVS simulators can be seen as hierarchical schedulers of events that activate the corresponding submodels. The schedules allow skipping periods of inactivity in the simulation. Nevertheless, explicit synchronization of the components is required, which involves a certain amount of overhead to be paid. Performance analysis of CD++ has been recently developed [Glinsky and Wainer 2002], and the study shows that the implementation of a real-time technique in CD++ is feasible.

We have implemented an extension of the simulation algorithms, which enable the system to execute DEVS models in Real-Time. In the following sections, we will present these extensions.

CD++ REAL-TIME SIMULATION

Virtual-time Simulation

The existing techniques in the CD++ toolkit employ a *virtual-time* approach. The methodology is useful for non-interactive simulation. This strategy advances the time disregarding any real clock attached to the simulation mechanism and periods of inactivity are skipped by the tool. In contrast to a *real-time* simulation, it is useless to connect inputs and outputs to the environment when the *virtual-time* simulation is performed, because the time in the simulation framework does not evolve at the same speed as within its surroundings.

In order to execute a simulation using the *virtual time* approach, CD++ maintains a variable in which the current *simulation time* is stored and updated. Again, note that this value is not linked at all to any physical clock. The update of that variable is performed by the simulator.

Simulation Mechanism

The simulation in CD++ is carried out by *Processors* that drive the simulation by exchanging messages. Two types of *Processors* exist:

1. *Simulators:* drive the simulation of atomic models, and
2. *Coordinators:* drive the execution of coupled components and coordinate the activities of all their dependant children.

A *simulator* object manages an associated *atomic* object, handling the execution of its δ_{int} (internal transition function), δ_{ext} (external transition function) and λ (output function). A *coordinator* object manages an associated coupled object.

Only one *root coordinator* exists in a simulation. It manages global aspects of the simulation. It is involved with the topmost-coupled component, which has the highest level in the model hierarchy. Moreover, the *root coordinator* maintains the global time, and it starts and stops the simulation process. Lastly, it receives the output results that must be sent to the environment.

Due to the hierarchical nature of the DEVS formalism, the message passing between simulators and coordinators usually consumes an important amount of time during the simulation.

The Flattened Simulation Technique

The overhead that results from the exchange of messages between processors could be minimized if the hierarchy is properly flattened. Therefore, the number of messages can be reduced accordingly [Kim et al., 2000]. It is important to conserve the usual model definition, execution, and the separation between *models* and *processors*.

Here, we introduce a *flattened coordinator* to provide a *flattened simulation technique* in CD++. This new *processor* is unique all across the processors' hierarchy and replaces all the usual *coordinators* and *simulators* existing in a hierarchical approach. Now, the *flattened coordinator* is in charge of different tasks that include the simulation of all the existing atomic components. Besides this, it carries out all the scheduling and port mapping among its children.

The following figure shows a sample model with a few components:

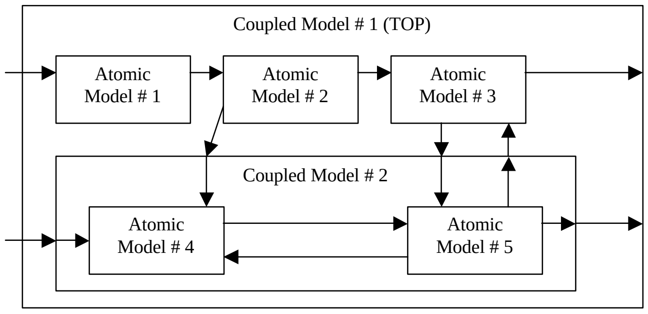

Figure 1. Sample model

The figure shows a sample model whose topmost component has three atomic submodels (*Atomic Models #1*, *#2* and *#3*) and one coupled model (*Coupled Model #2*). That inner-coupled component is formed by two atomic components (*Atomic Models #4* and *#5*).

The corresponding *model hierarchy* for the depicted sample is shown below:

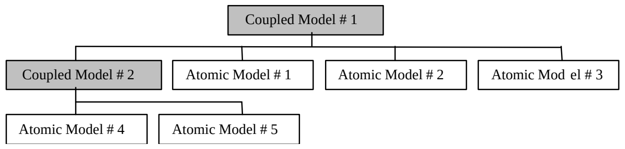

Figure 2. Hierarchical models' hierarchy

The processor hierarchy corresponding to this example is shown in the following figure.

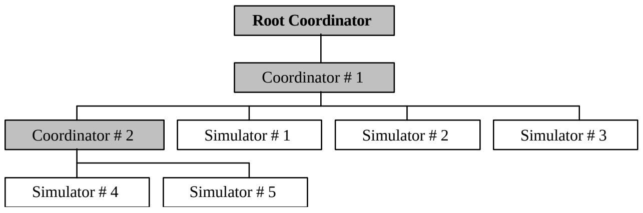

Figure 3. Processors' hierarchy (hierarchical approach)

As we can observe in *Figure 3*, whenever the *root coordinator* has to schedule an event to lowermost simulators (*Simulators #4* and *#5*) the overhead incurred by message passing can be considerable. The same phenomenon is produced if the *Simulator #5* sends an output through a port connected to *Simulator #3*. The number of intermediate coordinators can be arbitrarily high depending on the studied model.

If we simulate the model shown using the *flattened approach*, the resulting hierarchy is remarkably simplified and the overhead incurred by message passing is significantly reduced.

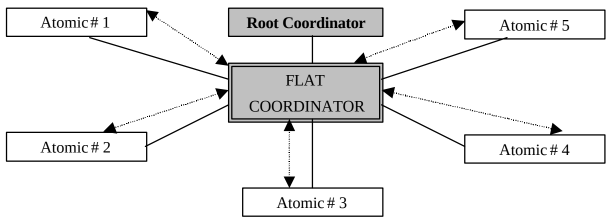

Figure 4. Processors' hierarchy (flattened approach)

In order to carry out the simulation properly using this new strategy, now the *flattened coordinator* has to store information concerning the atomic models he handles. Information about ports, links, time of next event, time of the last event processed as well as the queue of pending events must be saved.

Real-time Simulation

Modifications have been developed to allow real-time simulation in the CD++ toolkit. A real time system is defined as a system whose correctness depends not only on the logical results of computation, but also on the time at which the results are produced [Stankovic 1988]. If a system delivers the correct answer after a certain deadline, it could be regarded as an unsuccessful response. Consequently, a real-time simulator must handle events in a timeliness fashion where time constraints can be stated and validated. These new features would allow interaction between the simulator and the surrounding environment. Therefore, inputs could be received by ports connected to real input devices such as sensors, timers, thermometers or even data collected from human interaction. Similarly, outputs could be sent through output ports connected to devices such as motors, transducers, gears, valves or any other component.

The *root coordinator* manages the advance of time along the simulation. When the *virtual-time* approach was used, the messages were immediately generated by the *root coordinator* to initiate a new simulation cycle. Alternatively, when the *real-time* simulation is performed, the coordinator must wait until the physical time reaches the next event time to initiate the new cycle. A new simulation cycle can be started due to:

- ❑ The reception of an *external event*, or
- ❑ The consumption of time indicated by *ta(s)*

In the real-time extension of the toolkit, periods of inactivity are not skipped. The simulation process remains quiescent while these idle periods are being experienced. The *root-coordinator* expects the scheduled time to be reached and only then starts the new simulation cycle.

Typically, a model has to react to an external event within a given time to produce an output in order to solve a given problem. For this reason, a way to indicate a deadline time for an external event is provided in the real-time extension of the toolkit. When a model is executed, the simulator is able to check whether the deadlines are being met.

EXECUTION EXAMPLES

The testing phase included models generated with a synthetic generator [Glinsky and Wainer 2002] specially designed to measure the overhead incurred by the simulator. Different sizes, shapes and workloads have been used to produce models to test the new approach.

The following figure shows one of the simulated sample models:

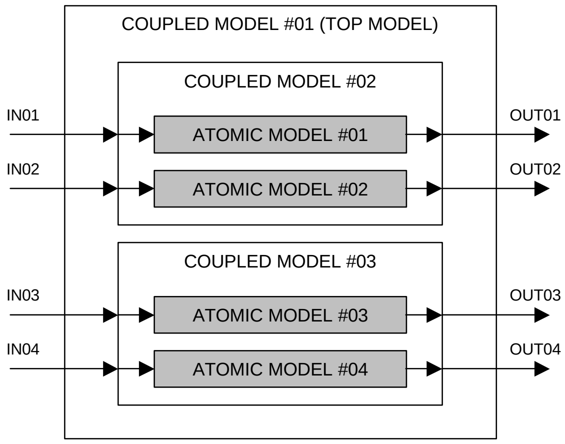

Figure 5. Simulated sample model

The model shown includes two coupled models, four input ports and four output ports. Both inner-coupled components are composed of two simple atomic models. Each atomic component is linked to the environment through one input port and one output port. The workload executed in each component varies from one atomic component to another.

Timeliness along a simulation is a substantial property in the real-time approach. When a model is being executed using this technique, it is usually important to check time constraints along the simulation. Particularly, the time at which an event has been completely processed is a meaningful measure of success.

Let us consider that the model must react to external events within a given deadline. We have provided a means to include this information in an **external event file**, which includes a number of events to be executed by the real-time environment. Each event arrives through a given *input port* with a given *value*. In addition, not only an *associated deadline* but also an *output port* must be indicated in the same event file. Thus, the simulator can check whether the physical time meets the associated deadline when sending an output through the associated port. Once the execution has finished, both successful and unsuccessful deadlines are stored for further study of the simulation process.

The following is a sample of such event file:

```
event_time    associated  in_port  out_port  value
              deadline

00:05:000     00:05:600   in01     out01     1
00:09:000     00:09:400   in02     out02     2
00:15:500     00:16:200   in03     out03     3
00:19:000     00:19:700   in04     out04     4
```

Figure 6. Sample event file in a real-time simulation

Figure 6 shows that the first event arrives at time *00:05:000* through the input port *in01*, and the expected output should be received through the output port *out01* before its deadline time, *00:05:600.* This states that the model must react to the given event in *600 milliseconds* or less.

The second event arrives at time *00:09:000* via the input port *in02*. The expected output should be received on port *out02* before time *00:09:400,* hence the model must react in *400 milliseconds* or less, and so on.

The simulator also keeps track of the *number of missed deadlines* and the *worst-case response time* throughout the execution, for further analysis. The *number of missed deadlines* represents the number of deadlines that have been missed along the entire execution of a model. On the other hand, the *worst-case response time* represents the maximum time between the arrival of an event and the output that the model produces in response, in the entire simulation process.

Once the simulation is completed, the obtained **output file** includes the *associated deadline* for each output and the *actual time* (*i.e.* wall-clock time) in which the outputs have been produced.

For each event, in the *result* column it is possible to obtain one of the following values:

Succeeded: **if** actual output time associated dea dline
Not succeeded: **if** actual output time > associated deadline

The next figure shows the corresponding output file for the executed sample model.

```
output    associated result     out_port value
time      deadline

00:05:500 00:05:600 succeeded  out01    1
00:09:300 00:09:400 succeeded  out02    2
00:17:100 00:16:200 not succ.  out03    3
00:19:900 00:19:700 not succ.  out04    4
```

Figure 7. Sample output file in a real-time simulation

In the previous example, the results informed after the four processed events are:

- Two events have been processed on time (“*succeeded”*)
- Two events have not been processed on time (“*not succeeded”*)
- The worst-case response time is *1600 milliseconds* (corresponds to the output produced at 00:17:100 whereas the external event was received at time 00:15:500)

Alarm-clock Sample

An alarm clock model [Jacques, 2001] has been used to analyze the real-time constraints under the new approach. The model has an important component of time and it is briefly presented here.

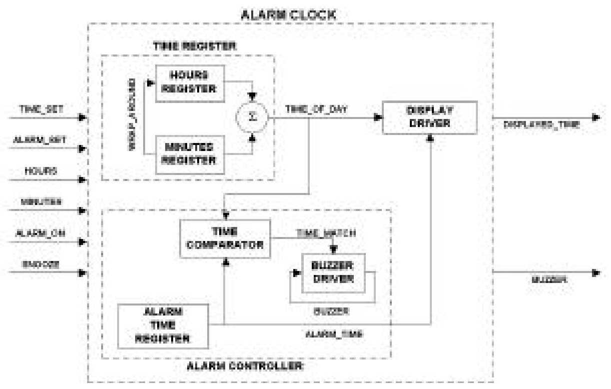

Figure 8. Alarm clock conceptual model [Jacques, 2001]

The model has three levels. The top level is the *ALARM CLOCK*. It has six input signals representing the push buttons and switch positions that exist in the real system. *TIME_SET* is used in combination with *HOURS* and *MINUTES* to set the time of day. *ALARM_SET* is used in conjunction with *HOURS* and *MINUTES* to set the desired alarm time. The buzzer will sound if ALARM_ON is set at that time. *SNOOZE* stops the buzzer for a period of 10 minutes after which the buzzer will automatically sound again if *ALARM_ON* is set. The model also has two outputs: *DISPLAY_TIME* represents the four-digit display while *BUZZER_ON* represents the output of the buzzer speaker.

The following is an excerpt from the output file produced by the simulation of the alarm clock.

```
actual       message    port          value
time         time

01:00:000    01:00:000  DISPLAY_TIME  00:01
02:00:000    02:00:000  DISPLAY_TIME  00:02
03:00:000    03:00:000  DISPLAY_TIME  00:03
...
30:00:000    30:00:000  DISPLAY_TIME  00:30
30:00:000    30:00:000  BUZZER_ON     1
31:00:000    31:00:000  DISPLAY_TIME  00:31
32:00:000    32:00:000  DISPLAY_TIME  00:32
```

Figure 9. Excerpt from the output file of the alarm clock

As time passes, the actual time is obtained through the *DISPLAY_TIME* port. Furthermore, the buzzer is turned on at *00:30* and this is notified through the *BUZZER_ON* port.

It is important to point out that actual output-times are equal to their corresponding message-times. This fact shows that delays are remarkably small all along the simulation. Therefore, such simulation can meet the deadlines imposed by the user.

Vending Machine Sample

Moreover, a vending machine model [Li, 2001] has been used for further analysis of the real-time extension in CD++.

The simulated vending machine is similar to the ones that exist in some cafeterias. Different items can be purchased by inserting sufficient amount of money and then selecting the appropriate button to dispense it. The machine returns the correct amount of change, keeps track of how many items have been dispensed and informs out-of-stock products to the customer.

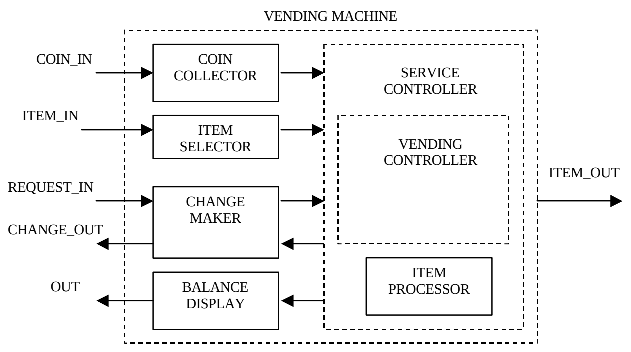

Figure 10. Vending machine conceptual model [Li, 2001]

The system includes several atomic components (a *coin collector*, an *item selector*, a *change maker*, a *balance display*, an *item processor* and others) and coupled components (a *service controller* and a *vending controller* inside of it).

The model has three input ports. Coins are inserted through the *COIN_IN* port, items are selected through the *ITEM_IN* port and change is requested through the *REQUEST_IN* port. The output ports are used as follows: *ITEM_OUT* is simulates the dispensed product, *OUT* resembles the balance display of the machine and *CHANGE_OUT* is used for the returned coins.

The following figure shows a sample event file, where a customer inserts different amounts of money and requests a particular item.

```
event      assoc.     in_port    assoc.     value
time       deadline              out_port

00:10:000 00:12:500  COIN_IN   OUT         0.25
00:15:000 00:17:500  COIN_IN   OUT         1.00
00:20:000 00:22:500  COIN_IN   OUT         0.25
00:25:000 00:30:000  ITEM_IN   ITEM_OUT    28
...
```

Figure 11. Sample event file - Vending machine

For instance, the first quarter is received through the *COIN_IN* port at time *00:10:000*, and the associated output is expected through the port OUT before *00:10:250*. Then a dollar (*1.00*) is received at time *00:15:000*, and so on. Finally, the item *28* is selected at time *00:25:000*.

```
actual       message      port          value
time         time

00:12:010    00:12:000    OUT           0.25
00:17:010    00:17:000    OUT           1.25
00:22:010    00:22:000    OUT           1.50
00:28:020    00:28:000    ITEM_OUT      28
00:30:010    00:30:000    OUT           0.00
...
```

Figure 12. Output file - Vending machine

Figure 12 shows the corresponding output file. The balance display is updated through the *OUT* port, two seconds after each coin is inserted. The item *28* is dispensed through the *ITEM_OUT* port at time *00:28:000*. Events are processed on time, and small differences can be observed between the *message time* and the *actual time* (wall-clock time) at which they have been produced.

Several executions have been performed using these and many other conceptual models. The results showed that the delays due to message passing are bounded, and the overhead is minimized when the flattened approach is used.

Timeliness is an essential and meaningful characteristic of real time simulations. In such simulations, whether a given deadline is met depends on different factors:

- *Overhead of the tool:* the execution of the simulation mechanism affects the overall performance. Usually, this overhead becomes larger as the size of the model increases, mainly because the time spent by exchanging messages among processors.

- *Workload in atomic components:* the more workload that has to be executed in internal and external transition functions, the more time that is needed to complete the execution of the corresponding code

- *Associated deadlines:* if the associated deadline for a given event is very tight, then it is not likely to be met. On the contrary, a loosened (relaxed) deadline is more likely to be met in a simulation.

The first factor, *overhead of the tool*, is intrinsically involved with the simulation process. It has to be minimized to allow a wide range of models to be executed properly using the real time simulation toolkit. The *flattened simulation approach* can achieve better results on performance because of the reduction of message exchange.

The second factor, *workload in atomic components*, varies from one model to another and depends on the characteristics of the models under execution.

Finally, the *associated deadline* influences the success of meeting a given deadline, and they are imposed by the user.

CONCLUSION

The real-time simulation has been successfully introduced to the CD++ toolkit. The proposed extension allows the interaction between the model and its surrounding environment. If such simulation is executed with the flattened technique, the communication delays incurred are minimized and therefore the real-time execution can be carried out properly. Several examples have been tested and analyzed using this technique and the results show that we are able to effectively develop real-time models for interactive simulations.

REFERENCES

Glinsky, E. and Wainer, G. 2002. "Performance analysis of DEVS environments". In *Proceedings of AI Simulation and Planning.* Lisbon, Portugal.

Jacques, Christian J.D. 2001. "Modelling and simulation of an alarm clock in CD++". Internal report, Dept. of SCE, Carleton University. Ottawa, ON.

Kim, K.; Kang W.; Sagong, B.; Seo, H. 1998. “Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One”. *Proceedings of the 33rd Annual Simulation Symposium.*

Li, Lidan, 2001. "Modelling and simulation of an vending machine in CD++". Internal report, Dept. of SCE, Carleton University. Ottawa, ON.

Martin, D.; McBayer, T.; Radhakrishnan, R.; Wilsey, P. 1997. *Time Warp Parallel Discrete Event Simulator.* University of Cincinnati.

Rodriguez, D. and Wainer, G. 1999. "New Extensions to the CD++ tool". In *Proceedings of SCS Summer Computer Simulation Conference*, Chicago, IL.

Stankovic J. 1988. “Misconceptions about real time computing: A serious problem for next generation systems.” *IEEE Computer*, Vol. 21, No. 10, pp. 10-19.

Troccoli, A. and Wainer, G. 2001. “Performance Analysis of Cellular Models with Parallel Cell-DEVS”. In *Proceedings of SCS Summer Computer Simulation Conference*, Orlando, FL.

Wainer, G.; Christen, G.; Dobniewski, A. 2001. “Defining DEVS models with the CD++ toolkit”. In *Proceedings of the European Simulation Symposium*, Marseille, France.

Zeigler, B.; Kim, T.; Praehofer, H. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems.* Academic Press.

AUTHOR BIOGRAPHIES

EZEQUIEL GLINSKY is a M. Sc. student in the Computer Sciences Department of the Universidad de Buenos Aires, Argentina. He is a Research and Teaching Assistant in the same department, and a member of the ParDEVS lab. He developed part of this work being a visiting research scholar at Carleton University. Glinsky has worked in the IT industry in Argentina for the past 7 years. Currently, he is an independent IT consultant.

GABRIEL WAINER received the M.Sc. (1993) and the Ph.D. degrees (1998, with highest honours) at the Universidad de Buenos Aires, Argentina, and DIAM/IUSPIM, Université d'Aix-Marseille III, France. He is Assistant Professor at the Systems and Computer Engineering, Carleton University (Ottawa, Canada). He was Assistant Professor at the Computer Sciences Dept. of the Universidad de Buenos Aires, Argentina, and a visiting research scholar at the Arizona Center of Integrated Modelling and Simulation (ACIMS, University of Arizona). He has published more than 60 articles in the field of operating systems, real-time systems and Discrete-Event simulation. He is author of a book on real-time systems and another on Discrete-Event simulation. He has been the PI of several research projects, and participated in different international research programs. Prof. Wainer is a member of the Board of Directors of The Society for Computer Simulation International (SCS). He is the coordinator of a group on DEVS standardization. He is Associate Editor of the Transactions of the SCS. He is also a Co-associate Director of the Ottawa Center of The McLeod Institute of Simulation Sciences.