Event-based Causality in Virtual Reality

Jean-Luc Lugrin, Paolo Libardi, Matthew J. Barnes, Mikael Le Bras and Marc Cavazza
School of Computing, University of Teesside,TS1 3BA, Middlesbrough,
J-l.lugrin@tees.ac.uk

Abstract - *In this paper, we describe such an environment, in which the user can be presented in real-time with an alternative range of consequences for a given interaction with virtual world objects, thus inducing various perceptions of causality. From a systemic perspective, we can adopt a pragmatic approach, inspired from Humian philosophy, which considers that causal relations are established by the user in response to certain event co-occurrences. To control causal perception in VR, the system comprises the following components: a visualisation engine, an Event Recognition System (together with its specific event formalism that supports event modification), and a causal engine as an Event Modification System, which selects co-occurring events on a rule-based manner. A search process, evaluates alternative consequences using semantic and spatio-contiguity information, such as comparison between candidate objects on which the action consequences should be applied. A first prototype has been fully implemented and is described together with an example real-time simulation.*

Keywords: Modeling and Simulation, Human Factors, Game Technologies, Interaction Techniques, Intelligent Virtual Environments, Causality,

1 Introduction

Causality is an essential aspect of our common sense understanding of the physical world. For that reason, causal perception has been considered as one of the main phenomena through which we perceive our everyday reality. This makes causality an interesting focus of experimentation when designing Virtual Reality System, including environments whose behaviour departs from our everyday experience. Virtual Reality Systems constitute one of the best examples of user-centred systems, in which subjects' experience derives from the intertwining of the system behaviour and their own. At the centre of user experience is the system's response to his own interaction with virtual world objects, which is mediated not only by individual objects behaviour, but by the integrated response of the environment as a whole. From the user's perspective, such a response is largely interpreted by attributing causal relations between user actions and system response. Yet, while causality has been widely st[1]udied in AI [12] and cognitive engineering, very little technical work has actually been dedicated to experimenting with the user's perception of causality in interactive settings. For instance, most of the multimedia experimental settings for investigating causal perception in cognitive psychology rely on simple, non-interactive computer animations [16]. This opens the possibility of creating virtual reality systems for experimenting with human perception of causality. Such systems have both a fundamental and practical interest. They can help better understanding of the perception of Virtual Reality, they can constitute research tools for Cognitive Psychology and finally, their technology can support new forms of interactive media, including in the Arts and Entertainment. In this paper, we describe research into the creation of virtual worlds in which causal laws could be modified, on a principled basis, so as to create alternative realities. The objective of this research is thus to create an experimental environment for causality. This takes the form of a virtual world in which every event can be intercepted in real-time, so that its consequences will be under the control of a "causal engine", rather than of a normal physical simulation. The causal engine operates by applying a set of transformation operators to the intercepted events, using a method inspired from search-based planning. These operators modify the effects of the intercepted events prior to their reactivation, thus resulting in the occurrence of alternative effects for the events considered. This makes it possible to create artificial co-occurrences from an original event, such co-occurrences inducing causal perception in the user. In the next sections, after reviewing some relevant conceptions of causality, we describe the architecture of our system, and the process for event interception and the recognition of high-level events. We introduce the search mechanism supported by the causal engine. Finally, we illustrate the behaviour of the system by detailing an example run.

2 Event-based causality

Causality is one of the best examples of an interdisciplinary concept in the field of Cognitive Science. There exists an abundant literature on causality in cognitive psychology, in Artificial Intelligence [12], and causality is still an active topic in contemporary Philosophy [6] [14]. This contributes to making causality a complex concept to discuss due to the intertwining of different notions derived from several research perspectives. Galavotti [6] proposes to distinguish "Property Causality" from "Event

Causality." Property Causality refers to population variables while Event Causality, also named Token Causality, refers to causality between single events. Event causality corresponds to common sense causal interpretation in the physical world, and is best exemplified by work on causal perception, from the historical experiments of Michotte [10] to the phenomena studied in developmental psychology [4]. Event causality is the kind of causality we'll be referring to in this paper. Further, because we'll be considering "real-time" causal perception, we will not discuss explanatory aspects. We will also be considering a user facing single events (single-event causality) rather than repeated experiments with co-variation measures [2]. However, an illustration of the difficulty of isolating one conception of causality is that concepts produced in the study of causal argumentation can bear relevance to the study of "physical" event causality, for instance co-occurrence, spatio-temporal contiguity, mechanistic arguments and the "no alternative" explanation [11]. This is why the experimental tool we are developing is trying to be as theory neutral as possible, and our approach is essentially pragmatic, almost in a Humian sense. Our system shall work by supporting the generation of non-standard event co-occurrences on a principled basis. Another important aspect is that these principles should be based on semantic descriptions of events and objects they involve, which goes beyond the spatio-temporal variations (e.g., gap and delay) traditionally used in the literature on causal perception [4]. In the long term, elements of a cognitive theory to be tested could be incorporated in those principles to support experimentation. Finally, recent contributions from philosophy have emphasised the role of agency in causality. Although agency has been associated both with property causality and event causality [13], it is of particular importance in our context, as we wish to test causal perception on interactive events initiated by the user.

3 Event intervention overview

The software architecture (Figure 1) shows the system's main operating cycle. It consists of the system intercepting events at a regular sampling rate, interpreting them at a semantic level, and modifying their default consequences, so as to create new co-occurrences in a rule-based manner. As such, it integrates two main components, which are an Event Recognition System and an Event Modification System.

- The Event Recognition System proposes an action formalism (Context Event), which is supported by a module developed on top of Unreal Engine Native Event System. We will refer to this module as the EIS module (Event Interception System). The EIS module represents a total of 25,000 lines of Unreal-script
- The Event Modification System provides a series of action alteration operators, called Macro-operators (Henceforth MOp). Our Causal Engine using a search-based mechanism applies those operators. The Causal Engine corresponds to 30.000 lines of C++ code.

We are using a state-of-the-art game engine, Unreal Tournament 2003, as a real-time visualisation system as well as a development environment. The use of game engines is now gaining great popularity in non-gaming Computing research [7], as these provide sophisticated visualisation and interaction features that enable the developer to concentrate on the writing of specific research

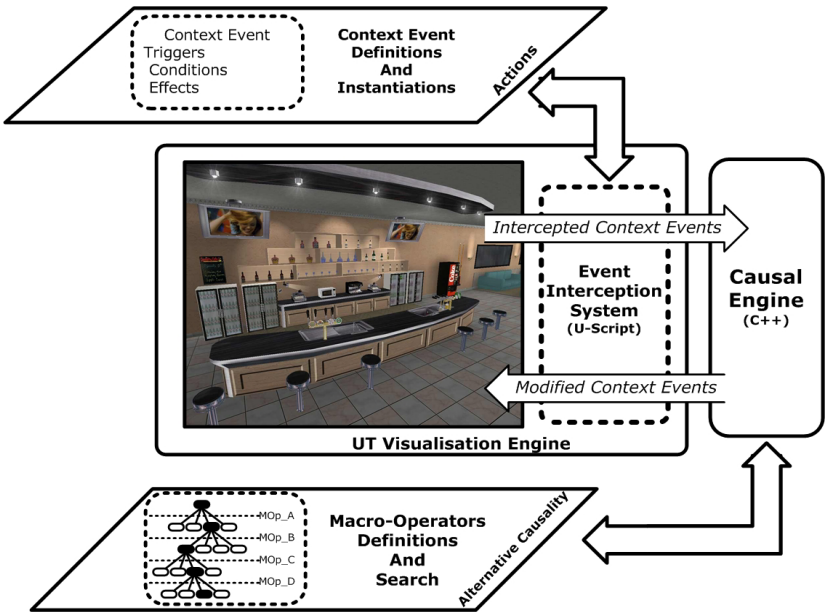

Figure 1: System Overview

software. In this context, UT 2003 includes a sophisticated event system supporting event interception, which we have used as the baseline layer to develop the Event Interception System. The EIS constantly intercepts low-level events, corresponding to objects' collision and motion, such as hitting, bumping, touching, entering volume. It then interprets these events and the objects they involve in terms of higher-level actions, called Context Events (CE). The CEs instantiated during a time sample are collected and then communicated to the Causal Engine via UDP sockets. After performing alterations on these events, the Causal Engine outputs a set of modified CEs, which are ready to be activated and have their effects triggered in the virtual world. As an overall result, the transformations operated by the causal engine on the CEs, representing actions in progress, produce new event co-occurrences in the virtual world, which association will induce causal impressions. Seen by the user, events occurring in the virtual world, including those corresponding to his own interaction with objects, generate different effects and consequences than those expected in the real physical world. Our system is designed to work both on desktop and immersive systems. In its immersive configuration, the hardware architecture consists of a SAS-Cube™, a four-sided PC-based CAVE™-like system powered by an ORAD PC cluster. We have extended several aspects of UT-CAVE developed at the University of Pittsburgh [7], in particular to support stereoscopic displays. Also, this configuration supports user interaction with the virtual world objects through a tracker-equipped joypad. This enables elementary 3D operations such as grasping, dropping and pushing objects. Other physical interactions involving motion, such as launching or throwing, have been directly associated with input keys on the joypad (figure 2).The detailed behaviour of these modules, as well as the working cycle of the whole system, are detailed in subsequent sections. The Context Event instantiation process is also illustrated and discussed in the next section.

Figure 2: Immersive Visualisation in the SAS-Cube

4 Event representation

In standard event-based virtual environments, behaviours tend to be encoded directly from low-level events. Event systems are generally derived from the low-level graphical event systems for collision detection (between objects, between objects and volumes). In this domain, the UT native event system proposes a large collection of events (called Native Events), such as `Bump (), Landed (), Hitwall (), Encroachedby () etc…` For each object class and/or states, "Event-Effect" relations are embedded in native event procedures (Call-Back System) associated to one or many effect procedures. When a Native-Event is detected by the visualisation engine, its effects' procedures are immediately instantiated and triggered, generating animations or object movements, as a response. Moreover, to obtain realistic animations, most of the virtual environments are coupled to powerful Physics/ Particle engine, as the case of the Karma™ engine used by the Unreal™ Engine. The figure 3 below shows an example of such hard-coded Event-Effect relations. This example describes what would be the behaviour of a Pint object *(Event Consumer)* if hit by another object *(Event Instigator)* using Unreal Physics/Particle Engine primitive: I.e. we simulate a glass explosion upon its collision with another actor (virtual Object). Such Ad hoc definition of causality (Cause-Consequence association) in a virtual

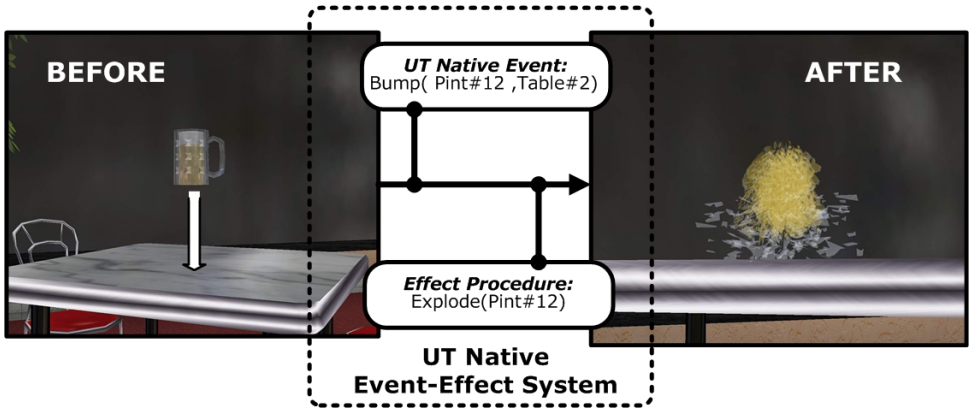

Figure 3: Cause-Consequence association within standard Event-based system

environment raises a certain number of problems. Firstly, as the "Event-Effect" relations are dispersed in the code, their identifications request expertise of the Environment and of its platform (Visualisation / Physics engines). Secondly, such hard-coded associations cannot support dynamic alterations based on principle. As a result, in standard Event-based VR systems, Causality is static, basic and hardly accessible. The Event Interception System we have developed on top of the UT Event-system, proposes to correct this lack of formalism and accessibility. Also it provides a complete interface between this formalism, where causal relations are expressed as Context Events, and the UT Visualisation / Physics engines. In our system, native low-level engine events are not directly linked to effect functions. The EIS module processes occurrences of the game engine's low-level native events, to produce intermediate-level events, called Basic Event such as `Hit (), Push (), Touch (), Press (), Enter (), Exit ()` etc. The instantiation of a BE (Basic Event) is based on additional conditions on the event (see Figure 4). For instance, the magnitude of the colliding object momentum in a colliding event can be used to instantiate a `Hit (?obj, ?surface)` from the system-level `Bump(?obj, ?surface)`. Basic Events constitute a base from which the derivation of higher-level events is possible. In our representation such high-level events are expressed as Context Events (CE). Context Events provide a proper semantic description of events, which clearly identifies actions and their consequences and therefore supports manipulation. Such high-level events explicitly encode default object behaviours in the environment. This module constitutes one of the innovative aspects of our approach, . in which an ontology for actions serves as a representation layer for the virtual world. Typically, a CE is represented using an action formalism inspired from those serving similar functions in planning and robotics (not withstanding the fact that they are used for creating rather than recognising actions), such as STRIPS [5], PDDL[11], or the operator representation in the SIPE system [15]. These representations originally describe operators responsible for transforming state of affairs in the world. They tend to be organised around pre-conditions, i.e. conditions that should be satisfied for them to take place and post-conditions, i.e. those world changes induced by their application. The formalism for CE comprises three main fields, which are analogue to the SIPE representation.

- The first field, called `triggers`, contains the basic events from which the CE can be recognised and which prompts instantiation of the corresponding CE. The CE Objects are assigned during this operation. For instance, in the figure 4, `(? Obj1, ?Obj2)` references the `Pint#1` and `Table#2` objects, on which the CE condition predicates will be operated.
- The `conditions` field is a formula testing property of the objects involved in such a CE. Conditions are expressed as conjunction of dynamic or/and static predicates. Certain predicates are targeting or comparing functional, visual, or physical object's states or properties. For instance, in Figure 4, the conditions specify physical properties of the CE's objects, namely that `?obj1` and `?obj2` are rigid and `?obj2` is harder than `?obj1`. The triggers and conditions fields govern the recognition part of the CE; once these fields can be instantiated by the CE recognition mechanism, a corresponding CE representation is created.
- The `effects` field corresponds to the consequence part of the CE and contains the effects to be applied to the objects affected by the CE. The effects are effect procedures usually generating animations that correspond to Object

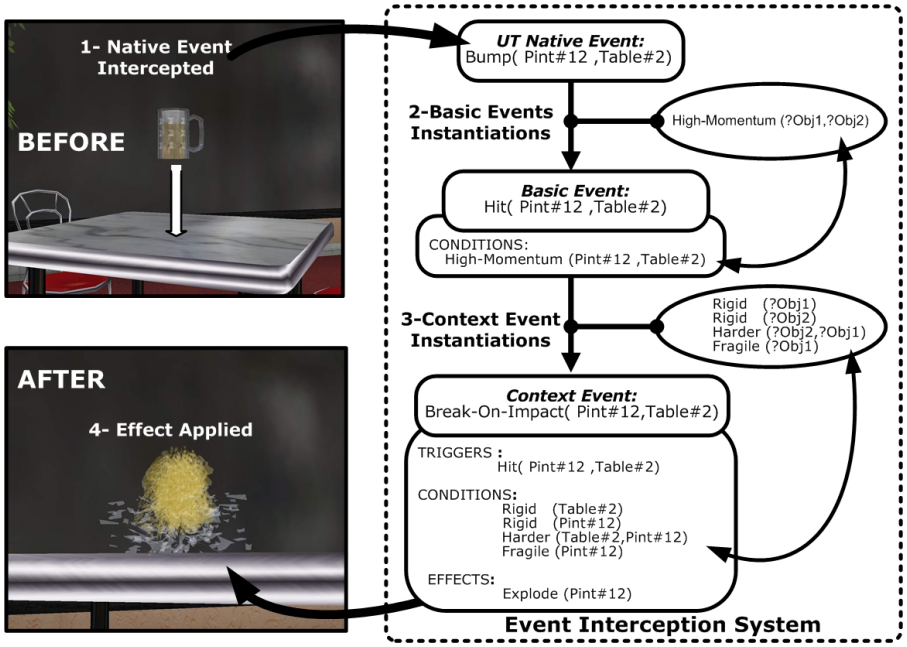

Figure 4: Instantiation of Context Event From Native Event

properties or state changes. Here, `Effects` will request the CE object: `?Obj1`, to explode.

A CE instance such as `Break-on-impact(Pint#12, Table#2)` should be interpreted as " `Pint#12` is going to break upon impact with `Table#2` ". That is why, context events are said to be instances of action in progress. In the next section, we describe how the combination of search and Macro-operators can generate new event-co-occurrences on a rule-based manner.

5 Event modification

The Causal Engine operates continuously through sampling cycles that are initiated by the occurrence of actions in the virtual world. Basically, the occurrence of events affecting world objects initiates a sampling cycle, during which the system recognises potential CEs and stores them while inhibiting their effects (it could be said that it "freezes" them). The causal engine then transforms these "frozen" CE, by altering their effects, before re-activating them. This re-activation then initiates a new sampling cycle. A view of the sampling cycle is presented on Figure 5.

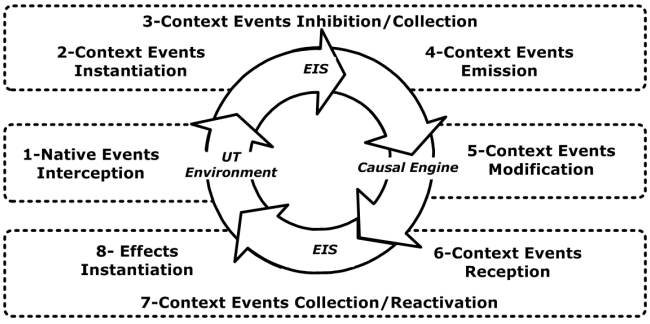

Figure 5: The system's event sampling cycle

The causal engine determines the modifications to apply to the world through a process inspired from search-based planning [1], in which the application of specific knowledge structures (Macro-Operators) is driven by heuristic search. Our causal engine searches forward in a best-first fashion by applying, at each iteration, different combinations of Macro-Operators. This operation is named MOp-Application, it generates new potential world states, each of them proposing a set of instantiated transformations. As explained in the next section, such transformations are weighted, and represented as Modified Context Events. MOps are described in terms of the classes of transformations they operate on CE's parameters, modifying CE's objects and/or effects. Examples of MOp classes include:

- `change-object`, which substitutes new objects to those originally affected by the CE's effects.
- `change-effects`, which modify the effects (consequences) of a CE
- `propagate-effects` extend the CE's effects to other semantically compatible objects in the environment
- `link-effects`, which relate one CE's effect to another's one

At the causal engine level, Macro-Operators are specific instances of the above classes. The causal engine is actually designed to handle multiple MOp-Applications on one or more CE, in order to produce complex effects or to operate at a global scene level (Figure 6).

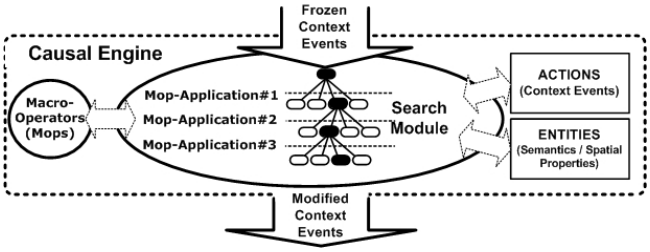

Figure 6: Macro-Operator search overview

The possibility of attributing a causal interpretation to a newly formed event co-occurrence is governed by cognitive principles (see e.g. [11]), which include spatio-temporal contiguity, but also semantic relations between events and the objects they involve. This is why, when modifying CE objects, Macro-Operators will precisely make use of both kinds of knowledge. Object compatibility functions are associated to each MOp class, which use spatial proximity information, as well as semantic information such as physical properties, shape, etc. For instance, objects which should break up as an effect of the CE could be replaced by similar, "breakable", objects. To accelerate the search, certain MOp class own specific evaluator functions. However, it should be noted that a individual MOp does not directly encode any specific causal relations. These would arise dynamically from the modifications induced by the MOp-Applications. As mentioned, the overall control mechanism of the causal engine consists of a heuristic search algorithm, which determines at each step which MOp transformations to apply. Further, in order to keep the world changes within certain boundaries, each time a particular Mop-Application is performed, creating a new world state, a cost function reflecting the extent of world transformation is calculated. At the end of the Mop-Application process, the search module evaluates and compares the heuristic. This threshold, experimentally determined, termed as the Maximum World Disruption. The search is terminated when the evaluation function exceeds this threshold. Then, the resulting set of modified context events is sent back to the EIS module for effects to be triggered in the

environment. However, if none of the generated world state values are higher than the threshold, this one is decremented by the highest cost. Then, the search routine is re-performed starting from a new world state presenting this highest value. In this sense, the search algorithm behaves as a cost-bound one. In the following simulation, we demonstrate from a single identical event, how the causal engine, by considering various values of threshold, generates alternative consequences.

6 Result

We have developed a test environment, the "Causality café", which recreates a familiar environment in which the user can interact with various objects and devices, such as glasses, bottles, taps, doors, fridges, etc (totalling around 300 reactive objects). The interactions with these objects are identified within a population of 60 context events (such as `Start-Fill-Container`, `Activate-Fluid-Source`, `Open-Container`, `Tilt-Container`, etc...). The test case we consider is that if a beer Bottle being grasped, then thrown by the user at a glass panel of a beer bottle refrigerator situated behind the bar. The default physical behaviour would consist of the panel breaking on the impact (top left on figure 7). However, intercepted by our causal engine, this default behaviour could be substituted by a large set of alternative consequences. For instance, low threshold values would give rise to alternative behaviours such as the impact of the bottle breaking another glass door. While higher values, usually provokes more surprising effects, like breaking bottles inside the fridge without actually penetrating the panel or breaking its glass. As illustrated in the figure 7, the number of MOp-Applications is proportional to the threshold. A low-level world disruption (low threshold) could force the search to exit after only one iteration (right side on the figure 7). Whereas high-level world disruption (high threshold) considerably increases the number of transformations applied, requesting multiple/consecutive MOp-Applications (a total of three Mop-Applications as shown in left side figure 7). At the end of a Mop-Application, the extent of world transformation proposed by each Mop combinations is evaluated and compared to the threshold. The following section details how a Macro-operator's instance measures the relevance of transformations it produces, and how that affects the search process.

As a starting event, when the bottle is colliding the glass door, a `Bump ()` event is detected by the Unreal^{Tm} engine. Using event recognition mechanisms previously described, the EIS module generates an instance of the CE Break-on-impact with the bottle and panel ID as parameters (like `Break-on-impact (Panel#1,Bottle#2))`. This instance is forwarded to the causal engine, when the event sample time has elapsed (usually 10-20 ms for only one user). The first action of the Causal engine when

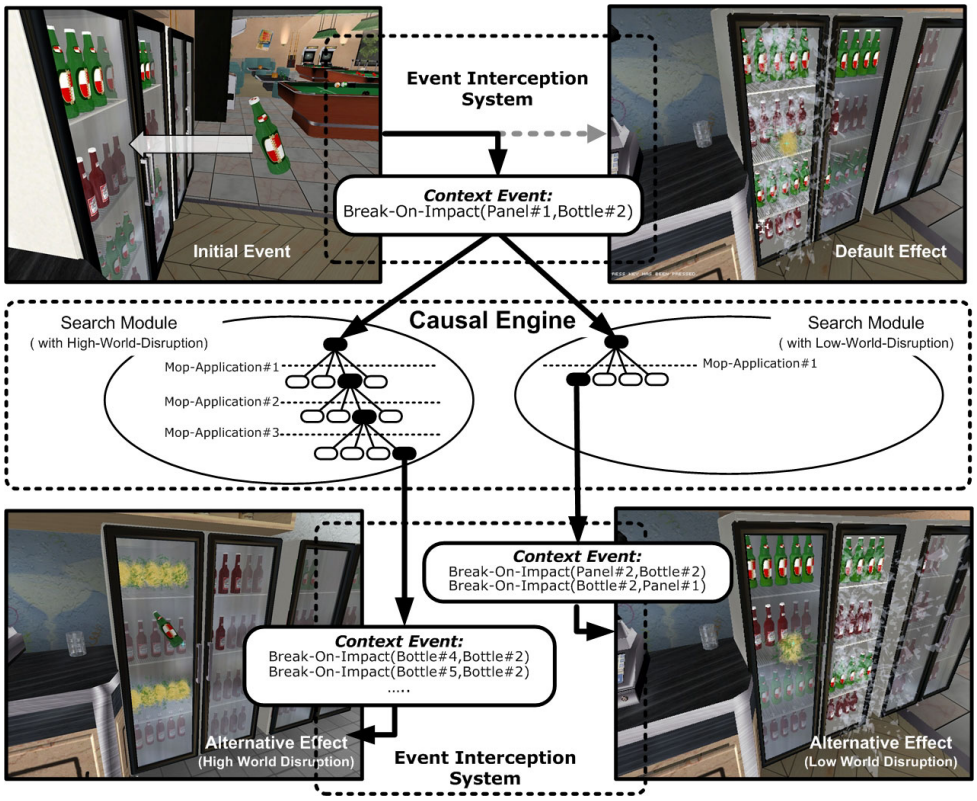

Figure 7: Generation of Co-occurrences by the Causal Engine.

receiving this CE instance is to generate a restricted list of "candidate" objects. This restriction is based on spatio-contiguity constraints calculated from the position of the original CE Objects. This first operation considerably reduces search execution time, while limiting the search-space to objects present in the user's field of view. The set of candidate objects, including the CE's original objects, and the set of Intercepted/Frozen CEs constitute our initial world state. In this example, we can list about sixty interactive objects, situated in a certain radius around the bottle / panel object (i.e. `Fridge#1, Fridge#2, FridgeDoor#1, FridgeDoor#2, Panel#1, Panel#2, Bottle#4, Bottle#5, Bottle #6,` etc…) Once an initial world state has been defined, the Causal Engine can start the search process by generating a first MOp-Application. A set of potential new world states is generated. Each new state proposes a new set of CE instances, called Modified CE. To obtain them, the causal engine has applied a combination of MOp instances to the initial world state, with different candidate objects each time. This procedure is terminated when all candidates have been processed once. In our simple example simulation, we use a basic set of 4 MOps on a population of candidates varying from 49 to 60 objects. As each Mop uses at least two objects, the MOp-Application will produce between 6 and 8 new world states. However, during this generation process, Macro-operators may associate a heuristic value, called degree of relevance (D.O.R) to each Modified CE. The transformations produced by a MOp instance are embodied in a `[Modified CE, D.O.R]` structure as highlighted by the pseudo-code below.

```
Trans [Modified CE, D.O.R] = Apply_Mop (Mop_Id, Frozen CE, Candidate Object)
```

The degree of relevance of a Modified CE corresponds to the multiplication of its semantic compatibility weight and its spatio-contiguity weight. The first value is produced by the compatibility functions of the Macro-Operators, based on the CE semantic constraints, contained in its condition fields. The second is a simple function computing and normalising the distance between the Frozen CE objects and Modified CE objects. Thus, the degree of relevance, expressed on a range of 0 to 1.0 can be simply formulised as below:

```
D.O.R = Semantic Compatibility Weight * Spatio-Contiguity Weight
```

This system provided a fast and principled mechanism to qualify the relevance of a transformation proposed by a macro-operator, in the frame of the action it is modifying (given by the Frozen CE manipulated). A low degree of relevance, close to zero, reflects a totally incompatible transformation. A high degree, close to one, that reveals a fully compatible transformation. For instance, we could consider different applications of a same Mop instance such as `Mop-substitute-action-object`, derived from the `change-object` MOp class. As suggested by its appellation, this MOp substitutes the object's instance, targeted by the frozen CE effect predicate `(Explode (Panel#1))`, by another object instance from the population of candidate objects. Within our example, it will first produce transformation instances such as:

- `Trans#1[break-on-impact (Fridge#1,Bottle#2), 0.0)]`
- `Trans#2[Break-on-impact Panel#2,Bottle#2), 0.0].`

Then, it will compute the Spatio-Contiguity weight regarding the distance of `Fridge#1` and `Panel#2` to `Panel#1`, which is present in the Frozen CE `(i.e. Break-on-impact (Panel#1,Bottle#2))`. The weights output would approximately be 0.9 for the `fridge#1`, parent of the `Panel#1`, and 0.7 for the `Panel#2`, positioned just beside Panel#1. After, the semantic compatibility weight of such transformations will be estimated by the Mop compatibility function. This function tests the applicability of a CE effect predicate on an object instance, considering the semantic constraint listed in the particular CE `Condition` field. It returns a normalised value, between 0 and 1.0, reflecting the number of predicates satisfied by the object instance tested. For instance, the `Fridge#1` that only validates the `rigid` property predicate of the `break-on-impact` CE will return a low value, close to zero (0.33). While, the `Panel#2`, that fully satisfies the `rigid (?Obj1)`, `Fragile(?Obj1)` and `Harder(?Obj2,?Obj1)` predicate, will return a value equal to one. As a result, we obtain:

- `Trans#1[break-on-impact(Fridge#1,Bottle#2),0.297]`
- `Trans#2[break-on-impact(Panel#2,Bottle#2), 0.7]`

The relevance of the `Trans#1` evaluated at 0.297 reflects an incompatible transformation, whereas the `Trans#2` relevance, which is valued at 0.7, appears clearly compatible within the context of the initial `Break-On-Impact` event. At the end of a MOp-Application, the search module compares the global relevance of each set of proposed transformations, to the targeted threshold . A cost-function sums the heuristic values specified for each solution. The first solution presenting a value closest to or superior to the threshold is selected. Consequently, the solution represents the optimal set of transformations in term of relevance. Thus, from a generic set of event modification operators (i.e. MOp), our system will always tend to provide an set of compatible transformations, while supporting mechanisms to extent or reduce their amplitudes.

7 Conclusion

We have presented a system, which supports the elicitation of causal impressions through the creation of event co-occurrences. The system can produce a vast number of modifications to "normal" consequences on a range of usual physical actions. These modifications are still determined by semantic considerations, hence ensuring believable, albeit non-standard, co-occurrences. The

combination of search and MOp application provides a powerful mechanism to generate associations and explore the space possible co-occurrences more systematically than through any a priori definition of co-occurrences. At the same time, there exists control mechanism which enable us to biase the search towards specific categories of effects. Overall, the system performance is in line with its initial design constraints, which imposed a response time in the order of 150-200 ms. for the moment, there is no indication as to how the system should scale-up to more complex environments. However, causal perception can only take place within the focus of attention of the user, which somehow suggests an upper bound on the environment's complexity. This research was originally driven by the creation of virtual reality experiences departing from our everyday experience, an approach we have termed alternative reality [3]. Moreover, such environments also have the potential to support various kinds of scientific experiments on causal perception, within a fully immersive and interactive setting, and as such could provide new tools for cognitive research

Acknowledgments

This work has been funded in part by the European Commission through the ALTERNE project (IST-38575). Marc Le Renard is thanked for providing the SAS-Cube Figure.

References

[1] Bonet, B. and Geffner, H. (2001). Planning as heuristic search. Artificial Intelligence: Special Issue on Heuristic Search, 129:5-33.

[2] Buehner, M. J. (2001). Inducing Causation: Covariation Assessment and the Assumption of Causal Power. In M. May & U. Oestermeier (Eds.), Interdisciplinary Perspectives on Causation. Norderstedt, Germany:Libri.

[3] Cavazza, M., Hartley, S., Lugrin, J.-L. and Le Bras, M. (2003). Alternative Reality: a New Platform for Digital Arts, ACM Virtual Reality Software and Technology Conference, Osaka, Japan, October 2003.

[4] Chaput, H. H. & Cohen, L. B. (2001). A model of infant causal perception and its development. Proceedings of the 23rd Annual Conference of the Cognitive Science Society

[5] Fikes, R. E. and Nilsson, N. J., 1971. STRIPS: a new approach to the, application of theorem proving to problem solving. Artificial Intelligence, 2 (3-4), pp.189-208

[6] Galavotti, M.C. (2001). Causality, Mechanisms and Manipulation. http://philsci-archive.pitt.edu/archive/00000132/

[7] Jacobson, J. and Hwang, Z. 2002 Unreal Tournament for Immersive Interactive Theater. Communications of the ACM, Vol. 45, 1, pp. 39-42.

[8] Lewis, M and Jacobson, Games Engines in Scientific Research. Communications of ACM, Vol. 45, No. I, January 2002. pp 27-31.

[9] McDermott, D. (1998) PDDL, the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control.

[10] Michotte, A. (1963). The perception of causality. New York: Basic Books. Translated from the French by T. R. and E.Miles.

[11] Oestermeier, U., & Hesse, F. W. (2001). Singular and general causal arguments. In J. D. Moore & K. Stenning (Eds.), Proceedings of the 23rd Annual Conference of the Cognitive Science Society (pp. 720-725). Mahwah, NJ: Erlbaum.

[12] Pearl, J. (2000). Causality: Models, Reasoning and Inference, Cambridge: Cambridge University Press

[13] Price, H and Menzies, P. (1993) Causation as a secondary quality. British Journal for the Philosophy of Science 44, pp. 187-203

[14] Price, H. (2001). Causation in the special sciences: the case for pragmatism. In Domenico Costantini, Maria Carla Galavotti and Patrick Suppes, eds., Stochastic Causality, CSLI Publications, pp. 103-120

[15] Wilkins, D. E. (1988). Causal reasoning in planning. Computational Intelligence, vol. 4, no. 4, pp. 373-380.

[16] Wolff, P. (2003). Direct causation in the linguistic coding and individuation of causal events. Cognition, 88, pp. 1-48

[17] Wolff, P., & Zettergren, M. (2002). A vector model of causal meaning. In Proceedings of the 23rd Annual Conference of the Cognitive Science Society Hillsdale, NJ: Erlbaum.