Catalá Bolós, A.; Pons, P.; Jaén Martínez, FJ.; Mocholi Agües, JA.; Navarro Martínez, EM. (2013). A meta-model for dataflow-based rules in smart environments: Evaluating user comprehension and performance. Science of Computer Programming. 78(10):1930-1950. doi:10.1016/j.scico.2012.06.010.

Alejandro Catala, Patricia Pons, Javier Jaen, Jose A. Mocholi, Elena Navarro, A meta-model for dataflow-based rules in smart environments: Evaluating user comprehension and performance, Science of Computer Programming, Volume 78, Issue 10, 1 October 2013, Pages 1930-1950, ISSN 0167-6423, http://dx.doi.org/10.1016/j.scico.2012.06.010.
(http://www.sciencedirect.com/science/article/pii/S0167642312001232)
Keywords: Ambient intelligence; Customization; Dataflow; Visual language; Rule; Event based; Non-expert programmer; Smart home

A Meta-Model for DataFlow-Based Rules in Smart Environments: Evaluating User Comprehension and Performance

Alejandro Catala[1], Patricia Pons[1], Javier Jaen[1], Jose A. Mocholi[1], Elena Navarro[2]

[1]Grupo ISSI, Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de Valencia
Camino de Vera s/n, 46022, Valencia, Spain
{acatala, ppons, fjaen, jmocholi}@dsic.upv.es

[2]Departamento de Sistemas Informáticos
Universidad de Castilla-La Mancha, 02071
Albacete, Spain
enavarro@dsi.uclm.es

Abstract

A considerable part of the behavior in smart environments relies on event-driven and rule specification. Rules are the mechanism most often used to enable user customization of the environment. However, the expressiveness of the rules available to users in editing and other tools is usually either limited or the available rule editing interfaces are not designed for end users with low skills in programming. This means we have to look for interaction techniques and new ways to define user customization rules. This paper describes a generic and flexible meta-model to support expressive rules enhanced with data flow expressions that will graphically support the definition of rules without writing code. An empirical study was conducted on the ease-of-understanding of the visual data flow expressions, which are the key elements in our rule proposal. The visual dataflow language was compared to its corresponding textual version in terms of comprehension and ease of learning by teenagers in exercises involving calculations, modifications, writing and detecting equivalences in expressions in both languages. Although the subjects had some previous experience in editing mathematical expressions on spreadsheets, the study found their performance with visual dataflows to be significantly better in calculation and modification exercises. This makes our dataflow approach a promising mechanism for expressing user-customized reactive behavior in AmI environments. The performance of the rule matching processor was validated by means of two stress tests to ensure that the meta-model approach adopted would be able to scale up with the number of types and instances in the space.

Keywords- Ambient Intelligence; Customization; Dataflow; Visual Language; Rule; Event based; Non-expert programmer; Smart Home.

1. Introduction

Many areas of computer research are now involved in developing the Ambient Intelligence (AmI) of the future [1][2], in which computers will actually disappear [3] and users will be offered intelligent, easy-to-use and effective services autonomously and transparently [4].

Much of this effort has been focused on the common objective of improving ubiquity, which has involved the development of architectures [5][6][7][8][9], discovery, aggregation, interpretation and service composition infrastructures capable of dealing with the heterogeneity of both the devices involved and the contextual information they supply [10][11], as well as providing adaptation mechanisms [12]. A lot of attention has also been given to the development of intelligence in subsystem sensing [13][14], mainly in activity sensing [15] and context sensing [16][17], with the idea of being able to provide services autonomously in accordance with user needs at different times.

Although proactivity and capacity for adaptation are absolutely necessary for the achievement of the concept of AmI, they entail a radical change in understanding the human-computer interaction, since to a certain extent they reduce the user's explicit interaction and increase automation. AmI is indeed user-centered, but in the search for implicit and autonomous interaction we are abandoning knowledge previously acquired in other types of studies on the user's role in configuration and adaptation.

Due not only to the heterogeneity of devices and appliances but also to the variability of situations and user preferences, even in specific application domains such as smart homes, it seems an almost impossible task at the present time to make infrastructures adapt themselves autonomously to real needs in all circumstances [18][19]. Not all situations can be easily learned, nor can system responses be correctly suggested for all users.

Of course we must not stop looking for more autonomous and intelligent systems, but neither must we overlook the knowledge provided by user personalization, as suggested by Weis [20], since it would probably facilitate the progressive development of AmI and would be closer to user requirements in exchange for obtaining greater explicit user interaction during the personalization stages.

Other authors have already seen the need for personalization in order to facilitate adaptation and configuration of systems in intelligent environments that contain all the possible user-based variability points [21]. However, in some of these studies, the tools are usually meant for designers, developers in their respective frameworks, or skilled end-users, but not really for users without or low programming skills [22][23]. In addition, in the proposals that include tools with interfaces for non-skilled users, more than personalization, they either allow rapid development of prototypes [24][25] or provide for the visual expression of simple rules, which, although powerful enough in certain cases, is somewhat limited in the application domain [26][27]. It would therefore be advantageous to explicitly involve the final users and allow them to provide part of the desired configuration for system adaptation, since they will be familiar with their own environment and their requirements. In this way the system will not have to depend exclusively on totally learned or automatic information, or information that has been encoded by developers and/or designers.

In the search of a generic tool to support personalization by non-expert programmers in the future AmI environments, we are combining the concept of rules with data flows in such a way that expressive rules can be specified and providing a language for reactive rules constructed around the concept of an ecosystem of entities. This enhancement in the level of expressiveness of the language to personalize smart environments must be manageable by users with either low skills in programming or at least certain digital literacy. Nowadays most young people are able to handle multiple computer applications from an early age, thus developing computational-thinking skills [28]. Since it is a generic and not a limited domain-specific language, it is highly advisable to evaluate the comprehension of the data flows involved as a formalism in the rule model, in such a way that we can say that the mechanism is certainly adequate for the task in spite of the inherent complexity of the language. Therefore, a first empirical study has been carried out with a group of adolescents. They have just been taught the basic computational concepts at school for the first time, and they represent the base of users with certain digital literacy as expected in the future inhabitants of future intelligent environments. Moreover, such a language must not compromise the performance of the systems, which have to select the right personalization behaviors to be enacted. In other words, as a secondary but not less important goal, the proposed expressiveness must be supported by effective enactment engines. Departing from a basic prototype [29] and the associate management middleware built to validate the viability of its construction, the performance of the implemented rule model was evaluated to demonstrate that the expressiveness of the proposed language is effectively supported at runtime.

The remainder of the paper is organized as follows: Section 2 describes some relevant related work oriented to provide customization by users. Section 3 presents ECAMI, describing the reactive rule model used as well as the meta-model behind it, and comparing it with previous proposals. Section 4 reports on the experimental evaluation performed on two aspects of our proposal. The first part presents a study on the use of data flows like those used in the rules by young non-expert programmers, and the second, consists of a verification that the rule engine performs asymptotically as expected for supporting a large number of entities in the AmI environment. Finally, a summary is given of the work's most important points, together with its achievements and a description of future research.

2. Related Work

This section describes a number of studies aimed at providing a mechanism for users to specify a behavior so that the system can be better adapted to their needs. Sohn and Dey [24] present a context-aware application prototyping tool which does not require coding and instead uses a graphical interface based on window controls, which allows context and devices to be collected and rules to be constructed (IF-THEN)

from them by taking only logical-relational operators and restrictions on types of complex conditions, with final testing in the devices or in a simulation environment. Even though it is really a prototyping support for developers, the technique is also considered suitable for end-users to modify context-aware applications.

The work by Beckmann and Dey in [27] incorporates an intuitive tangible interaction method for end-users to create control rules. The system relies on tangible pieces to represent rule conditions, which generally have real-world counterparts (e.g. a thermostat) using a metaphor for World-In-Miniature. The supported rule expressiveness allows the combination of up to three basic conditions and the invocation of predefined state changes in the environment.

A rule-based approach to control context-aware application behavior is used, and two rule builders are presented intended for end-users in the work by Zhang and Bruegge [26]. One is a window-based editor, focused on tabs, lists and checkbox controls, providing direct representation of rule concepts. The other is targeted at PDA devices and the user interface provides a user-friendly exemplar-based approach using puzzle-like pieces. In this, a conjunction of basic conditions can be expressed, and the action consists of mounting domain-specific icons together that represent service invocation or state changes. The user study reports that the PDA graphical user interface allowed users to define rules quicker with substantially less help. They made fewer mistakes and found it an improvement in terms of usability.

Dey *et al.* present in [30] a programming by demonstration context-aware prototyping environment intended for end-users. The system provides a series of data streams from different sensor inputs in such a way that users can select the input streams required for the behavior they want to express. It also specifies the actions to be executed. Machine learning algorithms interpret the annotated streams to determine the user's intention. Since end-users are perfectly familiar with their own activities and environments they are able to tell the environment how it should behave.

The work in [20] presents Nexel, a graphical tool for customizers based on [22], which provides a means of personalizing and fitting applications to specific needs as part of the development process for pervasive applications based on self-configurable components. The editor allows the user to specify the behavior to be used on the occurrence of the events that the devices trigger and can introduce bifurcation and iteration structure and instruction control sequences. This is done visually by dragging diverse sequentially connected components with different semantics to the specification area (e.g. *process, receive, post, loop*, etc.). However, the customization process is designed to support developers and customizers in development situations and not the end-users of the environment.

Considering end-users or low-experienced programming users should be allowed to specify the behavior of their environment, García-Herranz *et al*. present a rule-based agent system for ubiquitous systems in [31]. The system kernel supports powerful Event-Condition-Action (ECA) rule expressiveness as complex expressions can be written in their textual form and generic filters can be specified by means of wildcards. Several graphical user interfaces have been implemented on top to allow a range of users to control and program their smart spaces. One used a drag and drop desktop interface and another consisted of a simple GUI based on word tokens.

The recent work by Bonino *et al.* described in [32] sketches a visual interface that specifically targets non-skilled or low-technological skilled home-dwellers (i.e. non-expert users) based on a drag-and-drop metaphor. It relies on a rule grammar based on an IF-THEN structure with optional WHEN and OR-IF blocks for expressing conditions and rule alternatives. With an emphasis on providing strong visual cues and suggestions to facilitate incremental rule construction by end-users, the tool is intended to be implemented as a rich web application in the near future, but has not as yet been validated with real non-skilled users.

3. Rule-Based Behavior with ECAMI

Non-expert programmer oriented applications and systems are mostly event-based, as it has been indicated in [33]. An event-based approximation makes it easier to understand the rules that determine behavior, as shown by the results of certain experimental studies in [34][35]. Additionally, since behavior in

AmI is also event-driven, the rules are considered the ideal mechanism for this purpose [36]. In this section, the rule approach is described, presenting its main concerns, such as the simplifications of the environment, rule expressiveness and the meta-model, the dataflow approach to expressions and the basic mechanism for the enactment of rules.

3.1. A Simplified View of the Environment

Instead of taking a specific model in which concepts such as place, service, etc. are explicitly distinguished, in our work we chose a more generic and flexible approach which is not dependent on any concrete domain specific ontology to describe the elements of the smart environment. In this way, the backend of the rule definition and enactment systems is responsible for connecting details with the specific middleware used to manage the environment. We therefore based our approach on the assumption that the ambient intelligent space is simply composed of a set of typed entities that maintain a certain number of attribute-value pairs (i.e. properties) as well as the services the entities offer (i.e. actions). Figure 1 shows the UML class diagram of the conceptual model supporting entity-types as an aggregation of actions and properties that will eventually represent the services. The types of event that can be found in the environment are also specified in the model. An ecosystem is thus made up of entities which are instances of a definition of entity that specifies their behavior and the information available in terms of such actions and properties, abstracting any of the elements that inhabit the ecosystem. The ecosystem can therefore contain the definition of the types of entity for all the physical devices of interest, offering information in the form of properties that refer to values read directly by the physical sensors. Entity types can also be offered that

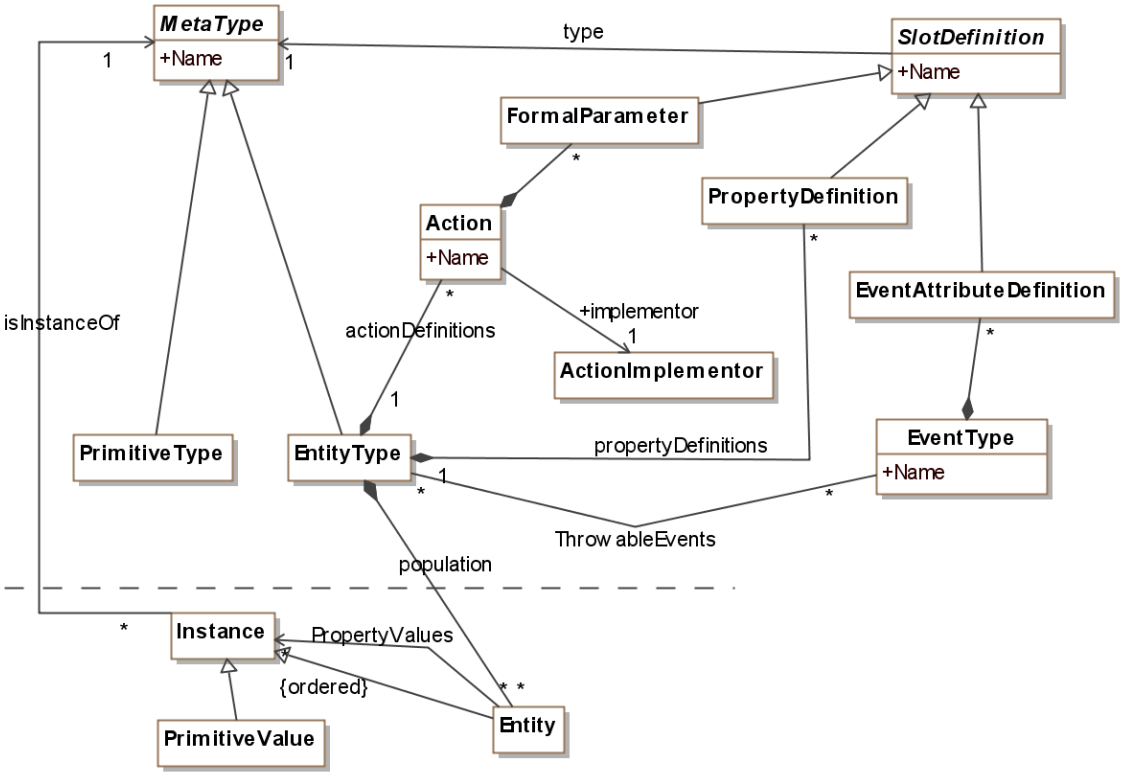


Figure 1. UML class diagram supporting a simplified entity-based vision of the environment.

represent virtual devices that help to extract specific ambient aspects as required, whose properties refer to values derived, aggregated, interpreted, or obtained from physical devices.

As long as the ecosystem exists, the entities are able to inject event occurrences into the system as the system develops. These occurrences correspond to a definition of a type of event. The event occurrences triggered by the entities will cause the state of the ecosystem to evolve as the specified rules are subsequently instantiated. The complete description of entity types, their actions, properties, event types and entities are sustained on a generic meta-model in such a way that the editor described in this work can cover a wide variety of scenarios in AmI spaces, provided that the editor backend is coupled to the specific middleware for the pervasive applications being used.

This vision is generic and flexible, as the specific concepts under consideration can change from one environment to another or from one deployment to another. Supporting this information with a meta-model provides additional benefits. For instance, maintenance and evolution are facilitated in the long term, as both seem important given the experimental nature of the field, in terms of introducing new sensors or new data-gathering abstractions. Moreover, the process of binding this meta-model with the specifics of the middleware can rely on an approach based on technological spaces as proposed in [37][38], instead of following an ad hoc approach, so that transformations and projections among the ontologies facilitate complete consistency and the effective use of tools in producing the system at a reduced cost, even taking more elaborated and refined intermediate domain-specific ontologies facilitating knowledge management [39].

3.2. Rule Expressiveness

To increase language expressiveness with respect to the previous approaches presented in Section 2, the scope of the rules under consideration was expanded to cover the entire ecosystem and were not linked to any specific entity. In this way the rules could be defined in a more natural way in which entity populations could intervene, instead of having to define a rule for each entity in a collection. This method can be useful for involving the same type of entity populations in the rules (e.g. devices of the same type such as temperature sensors), thus carrying out joint operations per type (e.g. changing the target temperature of several thermostats at once).

In this context a rule in our approach is formally defined as an ordered pair R = <P, Q>, where P is the antecedent and Q is the consequent. The antecedent is defined as P=(E, S, C) where E is the event definition that must be thrown by a source S (e.g. an entity), and C is the condition that must hold involving data (i.e. entity properties and event attributes) from E and S. The consequent is defined as Q=(T, F, O, {DP}) so that T is the target population, F is a condition that filters the entities within the target population to be affected by the outcome operation, O is the service operation on every target entity, and {DP} is a set of data processes that specify how the operation parameters are established prior to execution.

Operation O is a service invocation consisting of either the execution of an action or the assignment of a value to a property on the target. Consequently, a property assignment entails just a single data process in the set {DP}, which would specify how to compute the value to be established on the property. Similarly, if the operation is an invoked action then a variable set of data process structures would be needed, one for each action parameter.

The semantics is as follows: if an entity conforming to S throws an event occurrence of type E, and C holds, then the rule is triggered and instantiated. Operation O will be executed on those entities that belong to the target population T and meet the filter condition F. The operation parameters are established according to the data process specifications {DP}. In addition, the rules have an associated priority that allows the rule processor to rank them when establishing a prioritized order of execution. The structure of the formalized rule is as shown in Figure 2, and several example rules are given in Figure 3. For illustration purposes, we used a simple written English syntax at this point in the figures to clarify the conceptual

PRIORITY __
IF S:_____THROWS an EVENT E:_____
[AND C:_____]
THEN WITH T:_____
[SO THAT F:_____]
PERFORM O ({DP}):_____

Figure 2. Rule structure.

IF S: thermostat1 THROWS an EVENT E: TemperatureChanged
THEN
WITH T: monitor_display1
PERFORM O: Property Assignment (monitor_display1.temperatureF← thermostat1.temperature * (9/5) + 32)

_____

IF S: a person of ContextAware_Persons THROWS an EVENT E: Location_Changed
AND C: condition E.Destination = "Office"
THEN
WITH T: entity in Blinds
SO THAT F: T.Location = E.Destination
PERFORM Action Execution T.Open (level ← (25 * officeLightSensor.DiscreteLightLevel / 100))

Figure 3. Example of rules.

structure of a rule. We will later elaborate on the visual and textual languages that support the proposed rule model.

In our proposal, data processes are basically assignment expressions. In this way, the C and F conditions are also considered data processes that produce a Boolean outcome that one of the rules conditions must take. The language for these expressions embedded in the rule must at least support assignment, arithmetic and logical operators with the semantics of general-purpose languages. A grammar describing a textual language to define data processes with this expressiveness may be defined as follows:

```
Instr := Variable '<- ' Expr
Operand := Variable | Constant | Expr | Func
Operand_par := '(' Operand ')' | Operand
Operator := '+' | '-' | 'x' | 'AND' | 'OR' | 'NOT' | '>' | '<' | '='
Expr := Operand_par [ Operator Operand_par ]
Func := Function_name '(' Params ')'
Params := NIL | '(' Expr [, Expr]* ')'
Function_name := 'MAX' | 'MIN' | 'ABS' | ...
```

This grammar only includes a typical core of functions which has been used in the empirical evaluation below, but since the implementation of the rule proposal completely relies on a meta-model, extensibility is not difficult for developers allowing other additional functions like DISTANCE and NEAR_AT to be easily supported and made available to the users of the language. Figure 3 gives two examples of rules that can be defined in the proposed editor for a smart home environment. The upper example refers to a rule which would change the representation of the temperature to be displayed in a specific monitor. It also requires an extra conversion from Celsius to Fahrenheit of the actual source temperature obtained from the physical sensor. The lower example is a bit more complex as it involves source and target populations. The source in

the precondition responds to any person who enters the office, whereas the consequent specifies that all the blinds in the office will be opened according to the reading from the light sensor in the room.

3.3. Meta-Model Support for Rules

As occurred with entities and events which were defined flexibly in terms of a meta-model, rules and data processes are also conceptually described in terms of a meta-model supporting the previously discussed expressiveness. Figure 4 shows the UML class diagram supporting the formalized rule conceptual model.

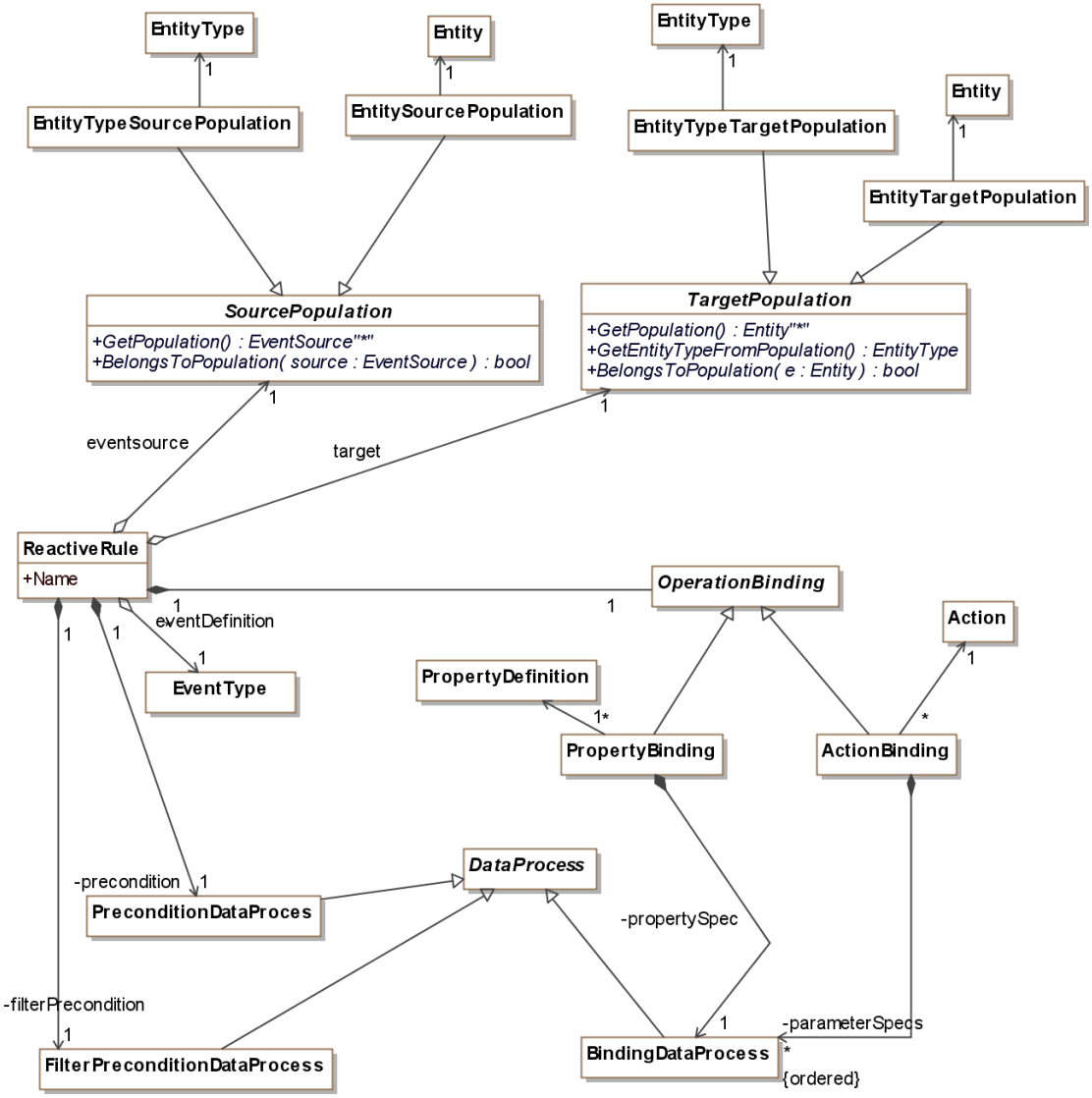


Figure 4. UML class diagram supporting rules.

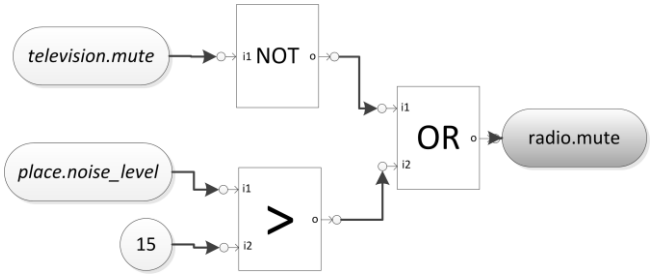
Firstly, a reactive rule consists of an *EventType* as defined in Section 3.1, a *TargetPopulation* and a *SourcePopulation*. Additionally, a precondition expression (*PreconditionDataProcess*) and a filter condition (*FilterPreconditionDataProcess*) are defined to establish the conditions that must hold to activate the rule and determine which target instances the consequent must be applied to. Finally, the operation-binding (*OperationBinding*) defines the operation that must be applied to the elements in the *TargetPopulation*, either a property assignment or an action invocation, and how the operation parameters must be computed by means of assignment expressions which are represented as data processes (*BindingDataProcess*).

The main concern is considering the specification of the event source as well as the target to be either a specific entity in the ecosystem (*EntitySourcePopulation* and *EntityTargetPopulation*) or a class of entities. In other words, indicating a particular entity (e.g. thermostat in bedroom), or perhaps a type of entity that will represent all the entities instantiated from it that are present and operational in the environment (e.g. all thermostats). The significance of representing source and/or target populations by means of entity-types (*EntityTypeSourcePopulation* and *EntityTypeTargetPopulation*) must therefore be kept in mind. If the source is defined as an entity-type, the rule will be instantiated if any entity of the type causes the expected event. On the other hand, if the rule target population is specified as an entity-type, the rule will be instantiated for every entity in the target population. This mechanism has advantages and allows the behaviour specification to be simplified when the exact entity is not known in advance or when there are various entities of the same type that must behave in exactly the same way (e.g. if all lights must be turned on or all blinds lowered in a room).

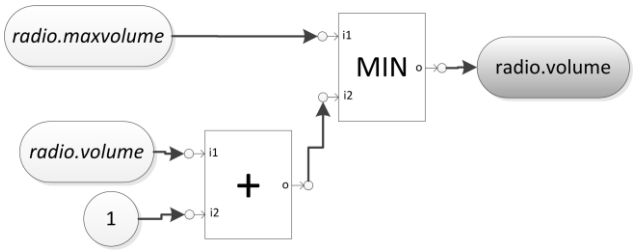
In other systems discussed in Section 2 the rules are specified in a simpler way, for example by not allowing any data transformation to be applied before being established as operation parameters, thus limiting expressiveness. Instead, our model uses data processes to define transformation operations on the available data (event attributes and properties of source entities among others), thus obtaining much more elaborate constructions. A mechanism is also provided to facilitate the involvement of the properties of any entity present in the ecosystem, with the aim of supporting rules with unlimited expressiveness. However, despite this expressive power, it is still possible to define simpler rules, such as those described in related studies, having no data processes at all in which the consequent of the rule uses values from the antecedent without performing any data transformation. The expressive capacity of our rule model is quite different to those in the above cited studies, particularly to those that contain a type of visual-based programming mechanism because they only support state actions without parameters or data transformations. However, we would like to aim at providing a useful editor with enhanced expressiveness for non-expert programmers in the future. This requires defining a language that can be easily oriented to user interfaces using visual and intuitive mechanisms. This is why a general but simple language based on data flow expressions has been chosen as candidate formalism to represent expressions in our language, and bound variables are limited to types with the role of event source or target. The present proposal is based on combining the simplicity of rule structures with the enormous expressiveness offered by data process structures in the form of dataflows, so that the resulting language is more graphically powerful and usable.

3.4. Dataflow expressions

The rule model presented enables a set of assignment expressions to be present in the rule specification. These expressions were generically represented in the meta-model by the *DataProcess* class in Figure 4 but nothing was discussed with regard to the actual syntactic representation of processes which could be represented in textual form, as in some previous examples. However, we are particularly interested in providing a rule editor that manages assignment expressions in a natural way and avoids writing code in the usual text-based form. This means devising an affordable mechanism to make programming easier, with a representation that facilitates the construction of a suitable editing tool. Although assignment expressions can be captured by different models (e.g. anyone based on operands and operators), the model considered is related to an explicit visual representation based on dataflows.



radio.mute ← place.noiselevel > 15 OR NOT (television.mute)
(a)



radio.volume ← MIN (radio.maxvolume, radio.volume + 1)
(b)

Figure 5. Sample expressions.

Figure 5 shows the type of expressions that can be supported. The expressions in the textual language look like many others used in computing and mathematics in general. The ones in the visual language have a tree-like organization, where the target variable to be assigned is on the right in grey, the operators are represented by boxes with inputs and outputs. The variables and constants involved in the process are conveniently connected by means of flows to the operators. The power of a visual paradigm relies on the fact that users can manipulate components as independent pieces, and simply join them without complicated interface controls.

The meta-model supporting the data processes is illustrated in Figure 6, in terms of processors and flows. To represent a dataflow-based expression, the model follows a graph-like approach, so that a data process (*DataProcess*) consists of a set of nodes (*DataProcessNode*) and a set of dataflow connecting node *Ports*. The nodes are specialized in a range of different supported node types, to essentially represent operators, value constants, or variables, and these nodes can be connected to others by means of dataflows (*DataFlow*) to form a complete expression. Our model contains an important type of node represented by the *DataProcessorInstance* class and allows the inclusion of operators in a flow, depicted as boxes in the figure samples. The definition of the specific operators available in the system is provided by the *DataProcessor* class, consisting of an operator implementation and a set of typed port definitions that will provide input parameters and an output result. As mentioned earlier, this is an important feature, as in this way the meta-model supports the extension of the language with new and valuable operators as needed.

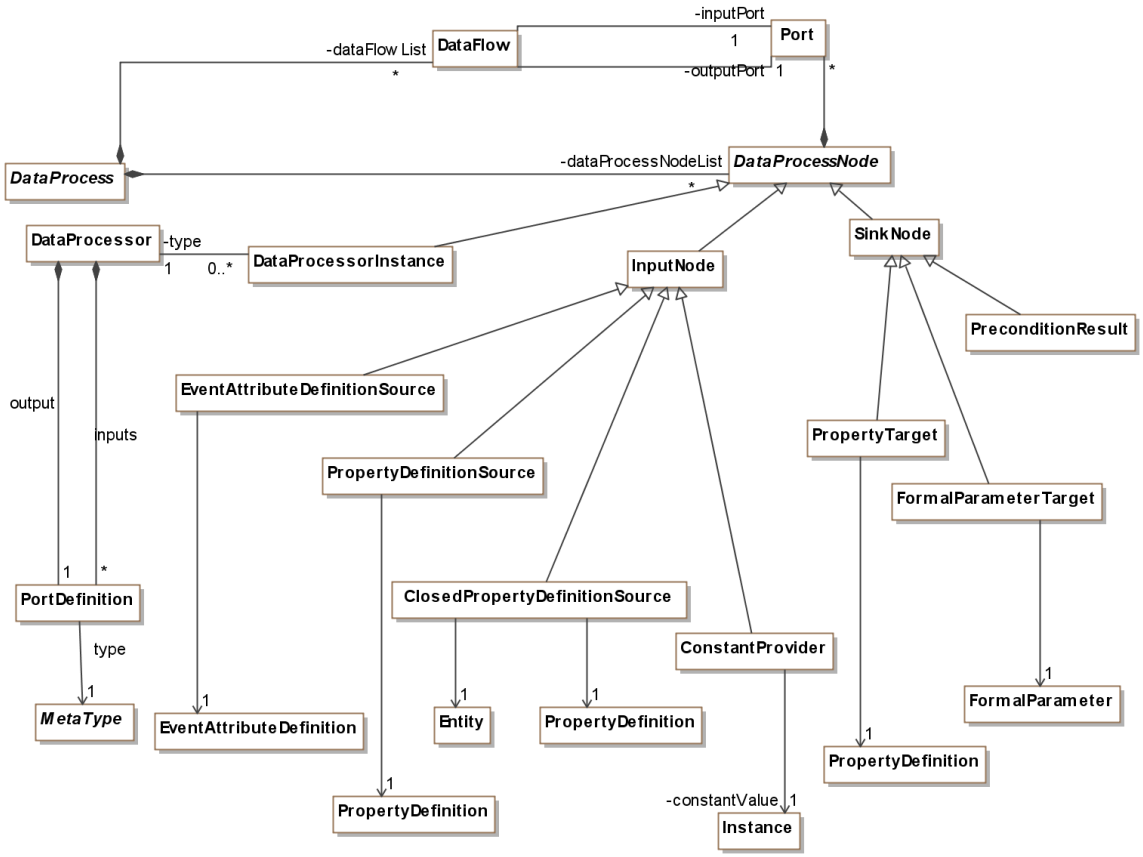


Figure 6. UML class diagram to support visual dataflows.

Besides the operators, there are other elements such as constants and variables which are also important in the definition of a data process. In this respect, the *ConstantProvider* node is an input node that provides the source of a pre-established value to be used in the data process (e.g. constant value "5"). This is to be directly connected to input ports. Variables can also be provided by three different types of nodes. If the value must be taken from a source's property or an event type's attribute involved in the rule, then the type of node is either a *PropertyDefinitionSource* or an *EventAttributeDefinitionSource*. Should it be required to take the value from a specific property of an entity other than the source or target, the *ClosedPropertyDefinitionSource* type node will facilitate such a specification. These different considerations with respect to the source element providing values to the variable are needed to support bound and free variables respectively. Essentially, the *PropertyDefinitionSource* nodes give access to properties as variables bound to the instantiation of S when triggering the rule. Alternatively, the *ClosedPropertyDefinitionSource* nodes allow the inclusion of property values from any entity as free variables.

Finally, there is a set of possible sink nodes that can be used according to the type of the assignment expression that is being created. A sink node represents the final node in which all the dataprocess computation converges and each expression must have one and only one. This node implicitly represents the assignment operation in the dataprocess. For instance, when a precondition expression is being expressed,

the sink node will be the *PreconditionResult* type. Alternatively, if the expression deals with an assignment to a target's property or target's action parameter, the sink node will be either the *PropertyTarget* or *FormalParameterTarget* type.

3.5. Rule Enactment

Now that the meta-model supporting the conceptual definition of rules has been presented the operational semantics that enables the enactment of rules needs to be defined. The rule engine that has been built relies on an event matching process on top of the meta-model described. This process is responsible for determining which rules must be instantiated and triggered according to the events occurring in the environment.

Given an event occurrence *ev*, the processor obtains the collection of rules whose event definition matches the one in *ev*. The *EventRuleMatch* class represents such a matching. Listings 1 and 2 show the specification of the matching process in pseudo-code, assuming that the rules are properly included during the initialization and the event occurrences are injected by the middleware as they occur.

Once the *EventRuleMatch* collection is constructed each rule in the collection is executed by evaluating the precondition data process and, if this is successful, by evaluating the existing transformation data processes to execute the action defined in the consequent.

However, along this basic operation of the rule processing, several additional inner aspects must be considered at the moment of executing rules to cope with conflicting situations. Given the expressiveness of the proposed language, there are two levels of conflicting behavior that may arise. Firstly, property-assignment conflicts in which several enabled rules attempt to modify the same entity property with different values, and secondly, method invocation conflicts in which several enabled rules invoke behavior on entities that could result in the system reaching some undesired state once the corresponding methods have been executed.

The first situation is typically the case of a rule whose operation is specified to set the temperature to 24 degrees while another activated rule is to set it to 21. It has been addressed in our system by implementing a priority based policy in which each rule is assigned a given priority. In the presence of a conflict such as the one just described, the system executes only the rule with the highest priority.

However the second situation is a bit more complex because it requires the formal specification of post-condition expressions that define the state that is reached by the system for each method execution. By having such a formal specification, logic reasoning engines could be used to detect inconsistencies or conflicting post-conditions as it has been done in the field of formal verification of programs. Although we have not included in this work this second approach, the model can be expanded to consider this in the future.

Along with these improvements, some additional reasoning process could be also included to support advanced useful features in real environments. For example, mechanisms to rollback rule actions would be interesting to allow users to undo the application of rules on an ad hoc basis. It requires the management of the property values over the time in order to be able to roll back reached states. In our system the case of property changes has been supported but the rollback of actions is not supported yet because it would require the complete specification of actions in terms of post-conditions, similarly to the issue presented above about conflicts. Another interesting functionality is the debugging of personalization rules. The middleware could analyze which rules are producing conflicts more often within a given interval since their activation, and this analysis could help users to detect faulty rules that would require some refinement. The combination of these features would also contribute to the capability of the middleware to automatically learn from the raised conflicts and/or exceptions of rule application (e.g. rollback situations) to improve the user experience by relying on temporal logics.

```
01: RuleEngine::MatchAndDispatch ()
02: BEGIN
03:    time <- System:Time();
04:    IF (emp.NewEventsAvailable) THEN
05:    BEGIN
           // performs matching (EventOccurrences against ReactiveRules; establishes the flags NewMatchesAvailable accordingly
07:        emp.Match ( time);
08:        IF (emp.NewMatchesAvailable) THEN
09:                FORALL erm: EventRuleMatch in emp.GetMatches()
10:                BEGIN
11:                        ExecuteRule (erm);
12:                ENDFOR;
13:                emp.ClearEventOccurrencesQueue ();

15:                // it empties the set and establishes the flag NewMatchesAvailable to FALSE
16:                emp.ClearMatches ();
17:        ENDIF;
18:    ENDIF;
19: END
```

Listing 1. Pseudo-code using the Event Matching Processor for the rule enactment.

```
01: EventMatchingProcessor::Match (gameTime:long)
02: BEGIN
03:    NewMatchesAvailable <- FALSE;

05:    FORALL eo: EventOccurrence in pendingEventOccurrences
06:        FORALL r:ReactiveRule in currentRuleSet
07:        // the private method MatchesRuleConditions checks for the agreement of rule conditions in both source and event against
the event occurrence
08:        IF (r.EventDefinition = eo.EventDefinition AND eo.RaisedBy BELONGS r.SOURCE.Population
               AND self.MatchesRuleConditions(r, eo)) THEN

10:                currentMatches.Add (new EventRuleMatch (eo, r));
11:                NewMatchesAvailable <- TRUE;
12:        ENDIF
13:        ENDFOR
14: ENDFOR
15: // clears the list of event occurrences and establishes the flag NewEventsAvailable to FALSE
16:  pendingEventOccurrences.Clear ();
17: END
```

Listing 2. Pseudo-code for the Match service of the event matching processor.

The build process of the *EventRuleMatch* collection is a key step that must be carefully designed if the performance of the overall rule execution process is not to be compromised. We will demonstrate in the experimental section of this paper that our engine is able to support an effective matching process under scalability conditions consisting of millions of entities in the ecosystem and thousands of entity types, way beyond the needs of most existing smart environments. To scale up to this problem size our matching process makes effective use of the meta-model relationships in which objects representing event-definitions, entity-types and entities that are part of a rule definition are conveniently linked with each other so that the

computational complexity of the matching process remains constant under some conditions that will be discussed in section 4.2.

The algorithm illustrated in Listing 1 is invoked by the middleware to perform the execution of rules in two steps. First the matching process obtains the *EventRuleMatch* collection (line 7), and the effective execution of the related operations is finally performed (line 11). Listing 2 details how the collection of correspondences between events and rules is calculated. The algorithm inspects the pending event occurrences (line 5) and determines which rules match for each one (lines 6-8).

3.6. Model Expressiveness Comparison

The related work described in section 2 does not usually state their meta-model explicitly. Thus, in order to highlight the most remarkable features of each approach and compare them, several features related to the meta-models and their expressiveness have been considered in Table 1 for a more systematic and convenient comparison.

Most of the related work relies on some sort of ECA rules as the basic underlying model. It is remarkable how many of them use Condition-Action (CA) rules, so that rules only include condition and actions. In those works, events are actually treated as conditions, and therefore there is not explicit distinction among these concepts (see the features *Underlying model* and *Trigger/Event* in Table 1).

Although conditions are supported throughout, there are some limitations and differences in expressiveness as the feature *Conditions* shows. Another important characteristic to be considered is related to variable support (see *Variable support* feature). This feature reports on whether the entities involved in conditions are only specific (i.e. particular instances already existing at design-time) or whether there is some support for bound variables. Bound variables are powerful because they allow writing expressions with variables (e.g. by means of types or wildcards) that are instantiated at run-time to one or several particular matching entities.

Some works support the construction of multiple excluding conditions that can be seen as different left-hand sides in a single rule (see the feature *LHS OR-ing*). Typically, it refers to syntactic constructions such as IF event/conditions OR IF event/conditions. They do not simply consist of using OR operators among conditions since the event can also vary. Not supporting this feature is not actually a hard limitation because several rules (i.e. one for each excluding event/condition) would be enough to cover all the cases. Indeed it could simplify the number of rules to be written.

Finally, the main differences among approaches are focused on which kind of actions and how many are supported in a single rule (see *Action* and *Composite actions* features). The most extended action model focuses on *state* actions. They typically refer to simple changes in the state of sensors or actuators (e.g. turn the light on/off or close the window). This type of actions also includes those requiring a single parameter although they essentially involve a simple state change (e.g. set target temperature to 23ºC). State actions cover a wide range of requirements while the environment is limited to state changes. However, there are some proposals that consider more complex action models involving either more parameters or non-trivial data-transformations on the parameters. In terms of editing tools, this increase in expressiveness is really challenging because data-transformations on several parameters have not been extensively studied, although it could make more powerful formalisms affordable in future research.

Our proposal presented in this paper basically has two primary differences with regards to other works. Firstly, there is support for bound variables but they are limited to the event sources and targets only. This decision was taken with the aim of being able to provide a manageable editor so that users do not have to deal with anything as a collection. Secondly, our model focuses on providing support for action parameters and data-transformations so that these elements may be edited without writing actual code.

Table 1. Comparison between related work

Work	Underlying model	Trigger or event	Conditions	Variable support	LHS OR-ing	Actions	Composite Actions	Users
Sohn & Dey [24]	CA rules	not present (event as condition)	three limited to relational operators	specific entities – no variables	yes	gradient actions (1 parameter with value levels)	single	suitable for end-users
Beckmann & Dey [27]	CA rules	not present (event as condition)	up to 3 pre-established discrete states	specific entities	no	state actions	simple	end users
Zhang & Bruegge[26]	CA rules	not present (event as condition)	conjunction of basic conditions	specific entities	no	state actions	multiple	programmers
Dey et al.[30]	Machine Learning (Dynamic Bayesian Networks)	implicit on levels of data streams	unlimited data streams	specific data streams	n/a	state actions	multiple	users who are not computer scientists
Weis et al. [20]	Signals & Slots	explicit (signal)	on signals	bound variables	program flow branching	slots valuation (~ state actions)	multiple	developers and skilled users
García-Herranz et al.[31]	ECA rules	explicit, although CA rules are supported	unrestricted but subexpressions unsupported (e.g. temp1 < temp2 + 5)	powerful bound variables (wildcards)	yes, or-ing for triggers	state actions	multiple	suitable for non-programmers end users using the magnet poetry metaphor
Bonino et al.[32]	ECA rules	explicit	conjunction of conditions (= operator)	specific entities	yes	state Actions	multiple	non-expert users (home inhabitants with little or no technological skills)
ECAMI	ECA rules	explicit	unlimited conditions, rich operators and mathematical expressions	bound variables (limited to event source or targets)	no	parameterized actions	single	low-experienced users (mathematical expressions)

4. Meta-Model Experimental Evaluation

Before developing the full middleware prototype and an editor supporting the proposed rule model, several aspects should be explored regarding the feasibility as well as the validity of the proposal. In this paper, two different aspects are considered. Firstly, as the visual programming approach is based on a dataflow-based formalism, which has not yet been considered in the context of end-user rule editors, the interest focuses on whether the dataflows are intelligible to non-expert programmers. Secondly, since we have opted for a meta-model approach rather than for a specific model, a point of verification is to check whether the rule model can effectively be operationalized and whether it is scalable with respect to the matching process, providing evidence that the implementation is feasible for further development. The studies concerning these aspects are presented in the following subsections.

4.1. User Study on DataFlow Comprehension

As mentioned above, one of the key elements for the specification of reactive behavior is the definition of data transformations by means of dataflow expressions. As end-users defining reactive behaviors in ambient intelligent environments with our rule model will have to effectively define these data transformation flows, it is important to evaluate the ease of understanding of the proposed visual dataflow language. However, unlike other studies that have done this on specific tools, using complex approximations including bifurcations and iterations, aimed at software engineering students with some experience in the subject [40], our study focused on basic structures suited to young non-programmers. The objective of the study was therefore to evaluate the ease of understanding, creation and evolution of expressions and ease of use of the system by this user segment. The comparison between our proposed visual approach and a textual language with a similar level of expressiveness was deliberately chosen because subjects are used to perceiving expressions such as those introduced in the previous examples in textual form in scientific subjects like Mathematics, Physics and Computer Science. In theory, this makes the comparison less favorable for the visual approach, but, as the experimental results will show, the visual properties of the proposed language have advantages over the textual language under certain conditions.

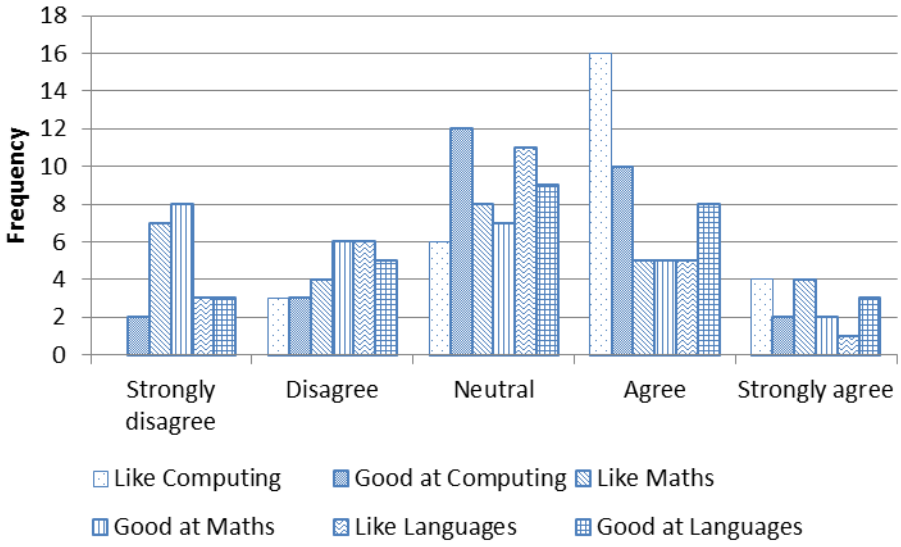


Figure 7. Demographic data on the preference and perceived performance in three main areas.

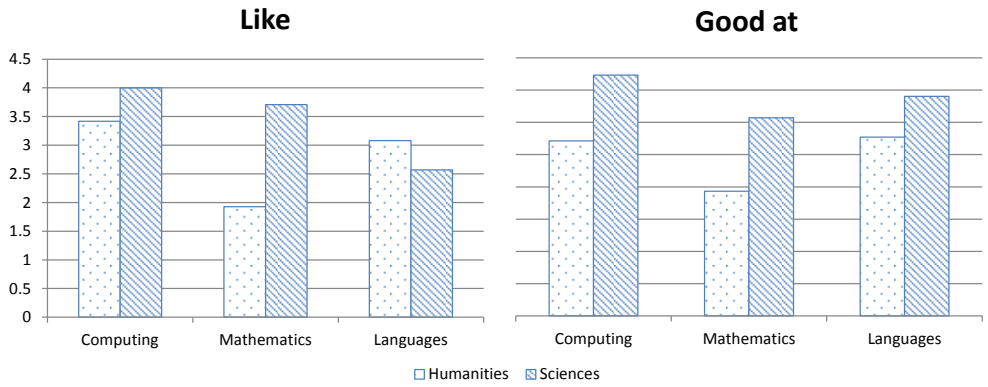


Figure 8. Average agreement for preference (like) and performance (good at) questions by curriculum

4.1.1. Participants

The participants were 36 fifth-year secondary school students (25 boys, 11 girls) who were taking a course in Computer Science as part of the curriculum. Most of them were 16 years old (M=16.39, SD=0.494) and were going to face real computational concepts for the first time. None reported prior programming experience except for basic spreadsheet formula definitions. Figure 7 shows the frequencies of the agreement degree on the participants liking for a subject and their performance in that subject. The students in general showed a real interest in Computer Science. A deeper analysis reveals that nineteen came from the Humanities stream whereas the rest came from the Sciences stream. The average agreement (1=strongly disagree, 5= strongly agree) by curriculum streams on those demographic questions is plotted in Figure 8. Those students coming from Humanities do like less mathematics and they consider themselves to perform worse than the students from the Sciences stream. Therefore, a reasonable expectation would be that students from Humanities could decrease the overall performance in exercises like those considered in the current study.

4.1.2. Materials

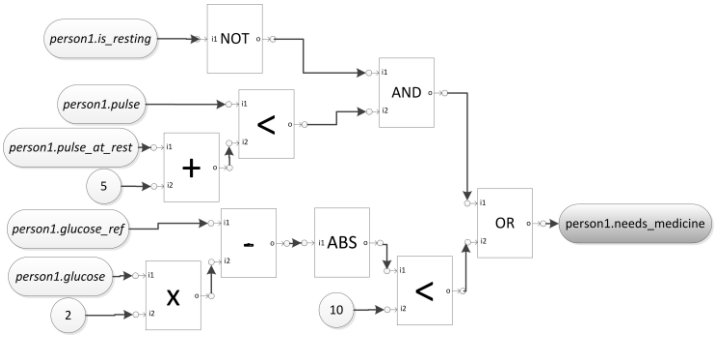
Teaching materials were created and adapted to the objectives of the course (i.e. introduction to programming). The class teacher devoted four sessions to the necessary basic concepts (variables, operators, assignment instructions, etc.) using the textual language of the grammar described above and the equivalent visual dataflow language, as shown in Figure 5. Both approximations were given the same importance and were accompanied by examples. Some sample exercises were carried out as a team.

The worksheet for the test was designed to explore the different aspects involved and to take students to the maximum skill level. Four exercise categories were considered. The *Compute* (C) category, typically presented an expression written in either the textual or visual language taught in class. The students were required to compute the outcome of the expression, given the values taken by the variables in the expression. This category was considered as it is an abstract mental process performed when working with expressions. When building and checking whether an expression is viable, people assign values to the variables and mentally test whether the computation matches the expected result. An example of this type of question is shown in Figure 9(a) and in visual language in Figure 9(b).

Assuming the following attribute values:
person1.glucose equals 4,
person1.glucose_ref equals 8,
person1.is_resting equals *Yes!*,
person1.pulse equals 8,
person1.pulse_at_rest equals 5.

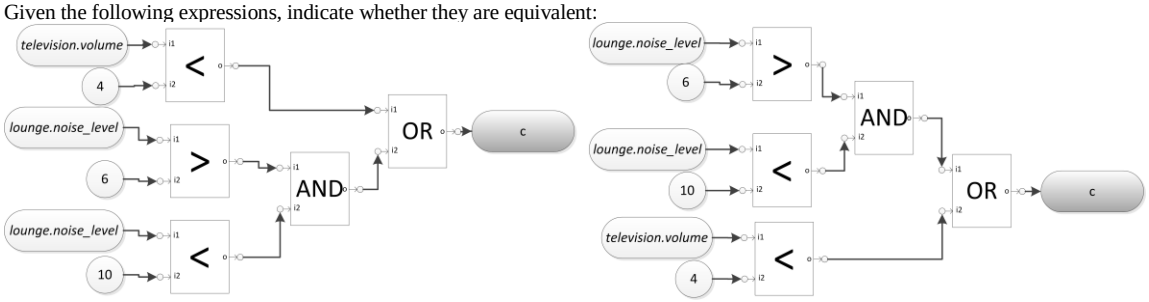
Calculate the result of the following expression:
person1.needs_medicine ←
NOT(person1.is_resting) AND (person1.
pulse < 5 + person1.pulse_at_rest) OR
ABS(person1.glucose_ref –
person1.glucose*2) < 10)

(a)



(b)

Given the following expressions, indicate whether they are equivalent:



(c)

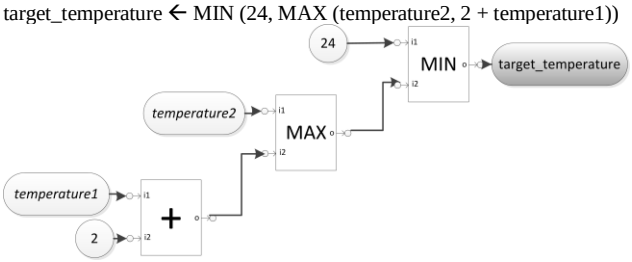
Given the following expressions, indicate whether they are equivalent:

c ← television.volume < 3 OR
lounge.noise_level > 5 AND
lounge.light_level < 12

c ← lounge.noise_level < 12 AND
lounge.noise_level > 5 OR
television.volume < 3

(d)

target_temperature ← MIN (24, MAX (temperature2, 2 + temperature1))



(e)

Figure 9. Sample exercises.

Another category was *Modify* (M). This consisted of facing the subject with an expression in either textual or visual form. Given a statement written in natural language, the participant was then required to detect and make the necessary changes to the original expression so that the proposed statement would be correctly specified. Since natural language is prone to ambiguity, statements were carefully written with a single interpretation. This category was included in the study because rules are usually subject to modifications until they work as expected. We therefore consider that for this lifecycle, it is important to be able to understand and "repair" a previously given version of an expression.

The *Equivalence* category (E), presented two expressions, both written in the same textual or visual language. The subject was required to indicate whether or not both were equivalent. This category is not only important during the lifecycle of expressions to detect mistakes and errors but also to evaluate the user's understanding of each language. Comparing two expressions requires more abstract reasoning, since values for testing expressions are not provided. An example of this type of question is shown in Figure 9(c) and (d).

Finally, a *Write* (W) category was also included in the study. The students were asked to write from scratch in a specific language an expression that matches a statement given in natural language. This statement was again ensured to be non-ambiguous.

These questions were of the type: "Write the following expression: *target_temperature* is the minimum between 24 and the maximum between *temperature2* and the addition of *temperature1* and *2*". The solutions to this question in the textual and visual languages are given by the expressions in Figure 9(e).

The worksheet included sixteen exercises, four in each category described above. Each exercise in one language had a corresponding exercise in the other, keeping the same expressional structure in order to ensure students were faced with expressions of similar difficulty in both languages. Four versions of the test were prepared, all of which had the same exercises but in a different order. Any two exercises from the same pair were not allowed to be contiguous and the categories were distributed throughout the test.

4.1.3. Method and Procedure

Both textual and visual languages were used to introduce primary concepts on programming in four one-hour introductory lectures. The students were tested after these teaching sessions. Worksheet versions were randomly distributed in equal numbers.

4.1.4. Task

The participants were provided with a worksheet in the test session and were asked to answer as many exercises as possible in the available time. They then answered several Likert-scaled questions on their subjective perception of each language on factors such as understandability, comprehension and learning effectiveness and some questions concerning which language version, textual or visual, the participant would prefer in the situations given in the questions.

4.1.5. Results and Discussion

The students were given a test containing exercises on the textual and visual languages they had seen in the teaching lessons. Table 2 shows the cross table for answers by languages and category. Due to the high difficulty and the number of exercises to be addressed in such a limited time without any advice to study the contents before the test session, we expected some unsolved exercises. There were 73 exercises in the textual language while 85 in the visual, but a Binomial test showed that the proportion of unsolved exercises in each language does not significantly differ from the hypothesized value of 50% (p-value=0.382). Overall 158 exercises remained unsolved out of a total of 576, resulting in 4.3 unsolved exercises per participant on average. It is interesting to observe that the values of the mean and the mode for the number of unsolved exercises are higher in the students from the Humanities stream (mean=4.94, mode=5) than those from

Sciences (mean=3.64, mode=3) as shown in Figure 10. This is an expected result because computing is mathematics after all, and therefore the worksheet required mathematical skills, which in the students from Humanities seemed to be less developed according to their answers to the demographic questions. Moreover, a deeper analysis of the distribution of the unsolved exercises per task reveals that (see Table 2) most unsolved questions were present in the Compute (52%) and Write (34%) tasks and that, in these cases, there are no significant differences (Binomial tests for C and W resulted in 0.728 and 0.392 respectively) in the number of unsolved exercises between the visual and textual languages. This means that the reason for leaving some exercises unsolved is not determined by the language but by the perceived difficulty of the tasks at hand given the limited available time to complete the experiment. In fact, this is confirmed by observing the subjective perceived difficulty of each task by the subjects (see column Overall for CQx questions in Table 3) who considered the Compute and Write tasks as the most difficult ones. All this suggests that the students tried to answer those exercises that considered more affordable given the limited time.

The answers to the exercises that were solved can be correct or incorrect. Unless the unsolved exercises, solved exercises did get an answer because students actually faced the given problem and made an effort trying to obtain the right solution, although they did not always success. Departing from these exercises we can assess the potential of the visual language in different tasks as the ones involved in the exercise categories.

Table 2. Summary of exercises by Answer, Language and Category.

Type				Answer			Type				Answer		
				Incorrect	**Correct**	**Unsolved**					**Incorrect**	**Correct**	**Unsolved**
C	**Lang**	**V**	Count	16	21	35	M	Lang	V	Count	15	38	19
			% within Lang	22,2%	29,2%	48,6%				% within Lang	20,8%	52,8%	26,4%
		T	Count	28	4	40			T	Count	31	35	6
			% within Lang	38,9%	5,6%	55,6%				% within Lang	43,1%	48,6%	8,3%
	Total		Count	44	25	75		Total		Count	46	73	25
			% within Lang	30,6%	17,4%	52,1%				% within Lang	31,9%	50,7%	17,4%
E	**Lang**	**V**	Count	10	59	3	W	Lang	V	Count	32	12	28
			% within Lang	13,9%	81,9%	4,2%				% within Lang	44,4%	16,7%	38,9%
		T	Count	7	59	6			T	Count	32	19	21
			% within Lang	9,7%	81,9%	8,3%				% within Lang	44,4%	26,4%	29,2%
	Total		Count	17	118	9		Total		Count	64	31	49
			% within Lang	11,8%	81,9%	6,3%				% within Lang	44,4%	21,5%	34,0%

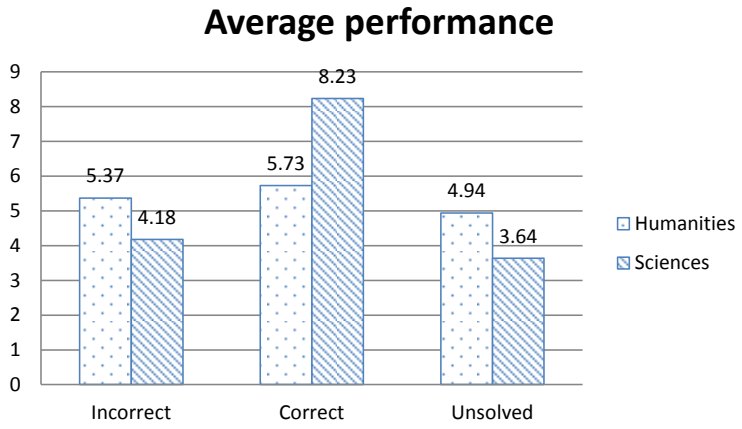


Figure 10. Average outcome per participant by curriculum stream.

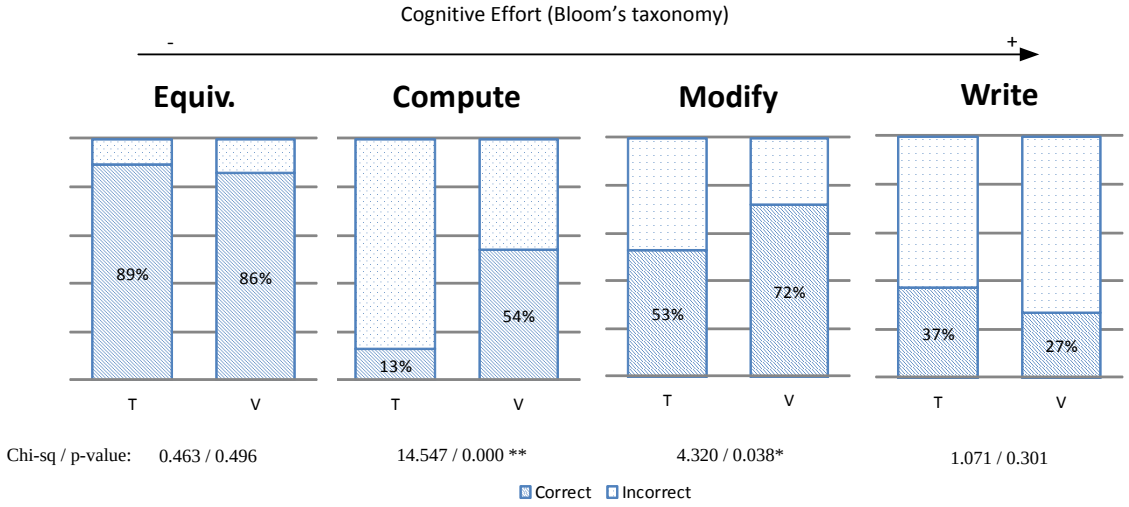


Figure 11. Answer performance per category.

Figure 11 gives the percentage of answer outcomes by category, arranged according to cognitive effort [41]. In the exercises with correct answers, the students performed better using the visual language in Category C and M, while the textual language gave better results in Category W. This tendency in favor of the textual language when writing expressions must be a consequence of the conditions of the experimental apparatus. We have to take into account that the visual language requires some sort of spatial arrangement of the elements and there was no tool support for this to help in the construction process, while in the textual condition requires writing expressions resembling the way the participants usually do (from left to right).

Nevertheless, only the comparison in Category C was found to be highly significant, while the comparison in Category M was of low significance, as shown by the corresponding Chi-squared tests for homogeneity of proportions. This means that the use of a certain language has an effect on the proportion of

correct answers in categories M and C (indicated in the plot by * and ** for significant and highly significant levels, respectively).

According to the overall results, the visual and textual languages performed similarly in tasks that were important for comparing and producing expressions. This is actually not a negative result, since the students had already seen textual expressions in mathematics and also in computer spreadsheets. The visual language performed better in exercises requiring computations and explicit evaluation of expressions. Although we expected tracing problems to be more difficult in the visual language because of having to follow data across parallel routes, the results indicate that in problems of the size dealt with in the present study this is not an issue and that textual representation is more prone to produce errors in calculations.

The worksheet also included a questionnaire to evaluate the user perception on several factors related to the understandability, comprehension and learning of the languages used on the worksheet. The first part of the questionnaire consisted of some general questions on learning the languages (GQx) and some basic questions to assess how difficult each category was (CQx). These questions used a 5-point Likert-scale in

Table 3 User Assessment Summary.

		Overall		V		T		
	Question	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**	**Mann. Whitney Test (Z / p-value)**
GQ1	I understand the meaning of the expressions	3.42	0.875	3.50	0.793	3.35	0.950	-0.629 / 0.530
GQ2	It was hard to learn how to use the language.	2.86	0.854	2.77	0.863	2.94	0.854	-0.773 / 0.440
GQ3	I consider easy to understand the different components of the language (operators, variables, etc.)	3.14	0.972	3.26	0.984	3.03	0.964	-0.979 / 0.327
CQ1	I find it difficult to determine whether two expressions are equivalent or not	2.69	0.129	3.00	0.913	2.41	0.907	-2.420/ 0.016 *
CQ2	I find it difficult to calculate expressions written in this language.	3.14	0.963	2.85	0.989	3.39	0.882	-2.202 / 0.028 *
CQ3	It is hard to modify existing expressions	2.53	1.002	2.57	0.978	2.63	1.006	-0.125 / 0.900
CQ4	I find it difficult to write expressions with the language	2.78	0.861	2.68	0.945	2.86	0.789	-1.071 / 0.284

Table 4. User preference summary.

	Question	**Textual (%)**	**Visual (%)**	**Test (Chi-sq / p-value)**
SQ1	It is easier to learn the language…	30	70	3.522 / 0.061
SQ2	I find it easier to understand the expressions written using…	32	68	2.909 / 0.088
SQ3	I understand the different elements better (operators, variables, etc.) in…	22	78	7348 / 0.007
SQ4	I learned the language faster …	26	74	5.261 / 0.022
SQ5	It is easier to calculate outcome values with the language…	17	83	9.783 / 0.002
SQ6	I am able to write expressions using the language more easily in…	46	54	0.182 / 0.670
SQ7	It is easier for me to know whether two expressions are equivalent using the language…	57	43	0.391 / 0.532
SQ8	When modifying an expression so that it represents a given statement I find out faster what needs to be modified using the language…	39	61	1.87 / 0.297

which “1” represented a strong disagreement with the given statement whereas “5” meant strongly agreement.

Table 3 shows the descriptive statistics and the Mann-Whitney tests to check whether there were differences on the agreement degree by language. Although the assessment of general questions show that learning and understanding the visual language was better rated than the textual representation, the tests showed that these differences are not important. However, for the questions about categories, the students perceived more difficulty in checking the equivalence of expressions in visual as well as in calculating in the textural language. With respect to the tasks involving modification and writing of expressions, they considered them equally difficult in both languages.

The second part consisted of single-choice questions concerned with which language, textual or visual, the participant would prefer in the situations given in the questions (SQx). This part of the questionnaire was answered by 24 students. It seems that the missing answers for this survey occurred as a result of being the last page of the worksheet and having no way to check whether the students answered it.

Table 4 summarizes the results. There is a marked preference for the visual dataflow language with regard to ease-of-use, ease of understanding, learning and use in calculations. This indicates that using a dataflow model seems to be a promising conceptual tool and should be seriously considered for non-programmers. Besides this empirical evidence, dataflows are an alternative to writing code since natural interfaces based on touch input are required in intelligent environments.

4.2. Meta-Model Performance Verification

The use of a meta-model, providing typed entities and reflection services, may facilitate the efficient matching of events and rules since relationships being defined could allow to effectively extract only those rules of interest for a given event occurrence. Thus, exploring the feasibility of the model operationalization is a first step to further middleware development and evaluation. Two synthetic tests have been specifically designed to show that the current implementation is scalable with respect to the time required to find the set of potential rules to be enacted (i.e. matching time) and therefore it is a promising foundation for a future parallel and distributed implementation. Essentially, the tests have only focused on the matching time and the study on how the typology of the source population (i.e. a specific entity vs. a population given as a type) can affect the matching time. Therefore, it must be understood as a preliminary verification before further and more extended evaluation in more realistic environments and conditions.

The first test focused on rules whose source population is specified as an entity-type (e.g. when any *window* gets open). Since rules expressed in this way refer to all the entities that belong to the population indicated by means of the type, the point to verify is that the matching time does not dramatically increase as either the number of entity-types or the number of entities within each entity-type grows. With this focus, the stress test considered a synthetic scenario in which entities injects events over the time to activate and trigger the rules, and the number of entities and the number of entity-type present in the system varied to see the evolution of the average matching time. The number of entity-types ranges between 1 and 40,000 whereas the number of entities within each entity-type is between 1 and 100. The number of rules available in the system is set to the number of entity-types present in the system at a given time. The aim was to demonstrate that the rule engine would be able to cope with a large number of entity instances in the environment while keeping the matching time constant for an event-rule pair when using rules related to populations. Figure 12 shows that matching time remains nearly constant when a maximum number of four million entity instances, which is the full workload tested (40,000 x 100).

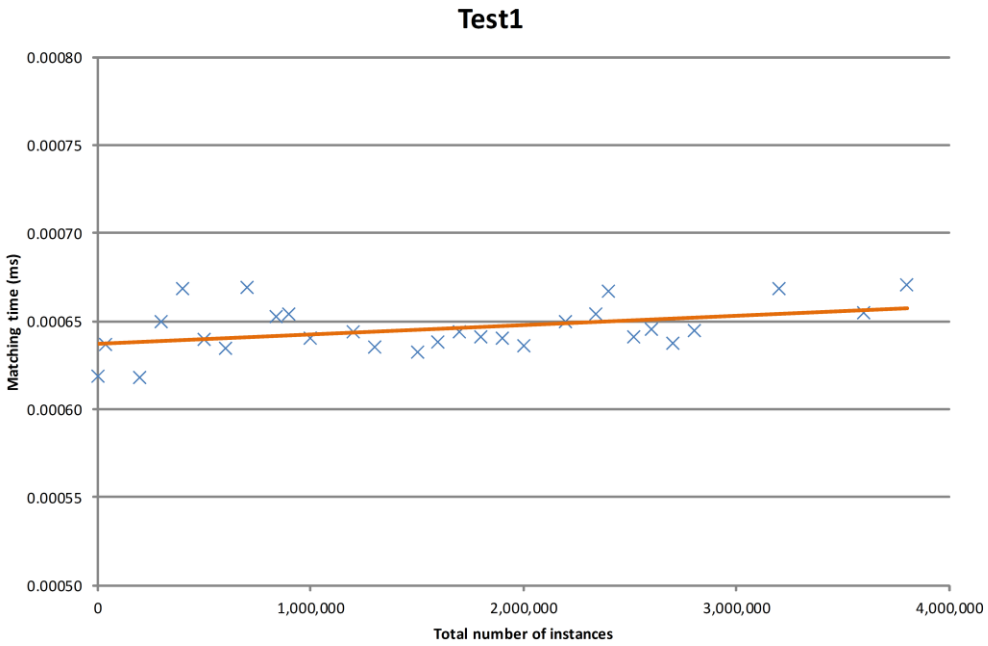


Figure 12. Matching time vs. total number of entity instances under the conditions of Test 1.

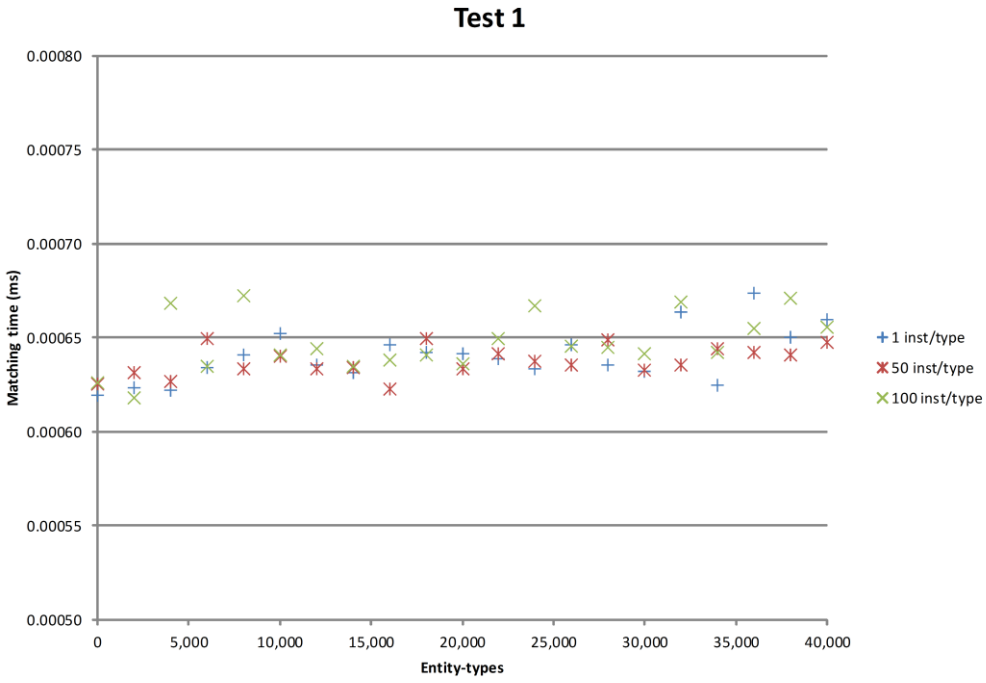


Figure 13. Matching time vs. the number of entity-types in the conditions of Test 1.

Moreover, as can be seen in Figure 13, the matching time is not affected by the number of entity-types in the ecosystem. It shows that this constant behavior is consistent no matter the available instances are spread over a reduced number of entity types (i.e. homogenous) or extended across a high number of entity types (i.e. heterogeneous). As confirmed by the experimental results the meta-model reflection feature associating rules with event-types provides a retrieval process for the set of rules associated to a given event-definition at a constant time.

Given that the current implementation is suitable to cope with event sources expressed as population by means of types no matter the size of the entities present, a second essential test would consist of exploring the matching time when using rules whose event source is defined over specific entities. In this particular case, every rule only refers to an event thrown by a concrete entity. Thus, in this second stress test the matching time has to be analyzed with respect to the variation of the number of rules defined for each pair consisting of an event-type and entity.

For each pair <E, S>, i.e. each pair event-type and source entity, a number of rules R were created (R=1, 5, 10). In this way, when an entity throws a given event occurrence, a set of rules will be triggered, since there are several rules that match the same pair E and S. As the total amount of rules depends on the number of event-types present in the system, it was varied between 1 and 2000, in order to explore how the matching time evolves as the set of matching rules increases. In addition, to show that if the relationships present in the meta-model are exploited then a set of efficient indexes can be built so that the matching time remains constant no matter the number of event types considered, two versions of the event-matching processor have been compared (optimized and non-optimized).

The first version was implemented without taking into account the valuable information contained in the meta-model of the rule specification, which needs to examine the whole rule set regardless of the current event occurrence or the entity that raised it. The second version, however, considered the construction of indexes on the rule set based on the relationships available at the meta-model level, so that the processor was able to effectively determine the rule set of interest for a given event occurrence. The construction of the index is performed once at the beginning of the simulation and it requires the inspection of the all the specified rules. Indeed, in a more realistic dynamic environment, there would be an extra cost to maintain the index structure updated, but the basic cost only implies the removing or insertion of single entries in a couple of hash tables performed in constant time when new rules are added or removed.

Figure 14 and Figure 15 plot the matching time in the above-described conditions for the non-optimized and optimized versions, respectively. As Figure 14 shows, the matching time is linear up to the number of event-types (where |rules| = R * |E|* 10), as the set of rules must be inspected in the non-optimized version. The optimized version obtained nearly constant matching times, as the optimizations are able to provide the appropriate rule set without any extra cost (see Figure 15).

The previous experiments show that the matching process scales with the number of entities, entity-types and rules per event-type even when the implementation is still centralized in only a computing node. This suggest that the system scalability in terms of the obtained throughput (i.e. number of events processed per unit of time) is a matter of the number of available parallel processing nodes (threads, cores or devices) that are used so that the matched rules may be executed in parallel. Therefore, a parallel future implementation of the processor would be promising to cope with other typical issues such as fault tolerance, workload distribution, low power and low memory computing nodes.

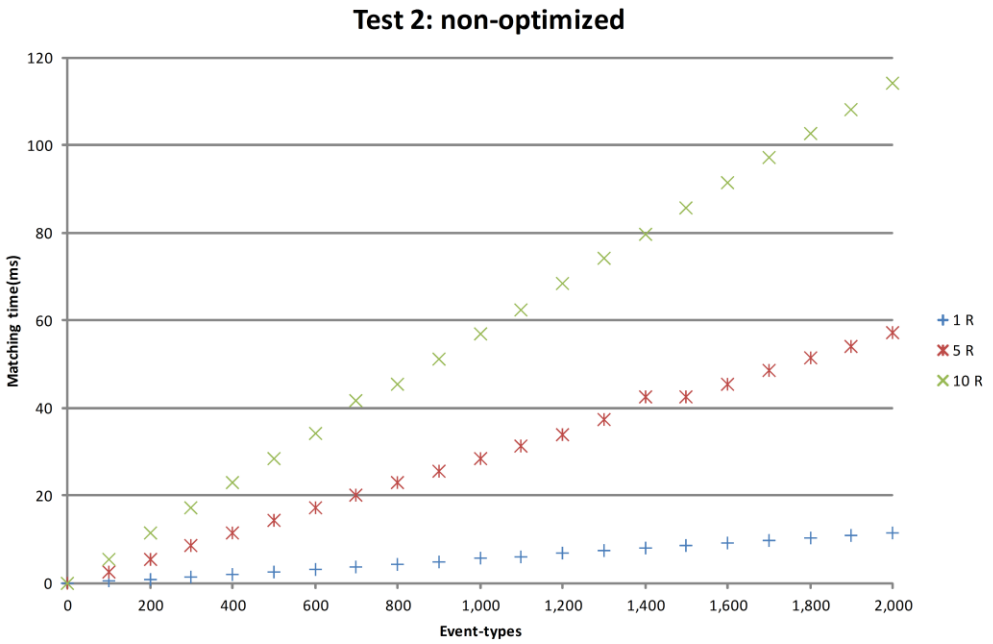


Figure 14. Matching time vs. event-types with repetitions under non-optimized conditions in Test 2.

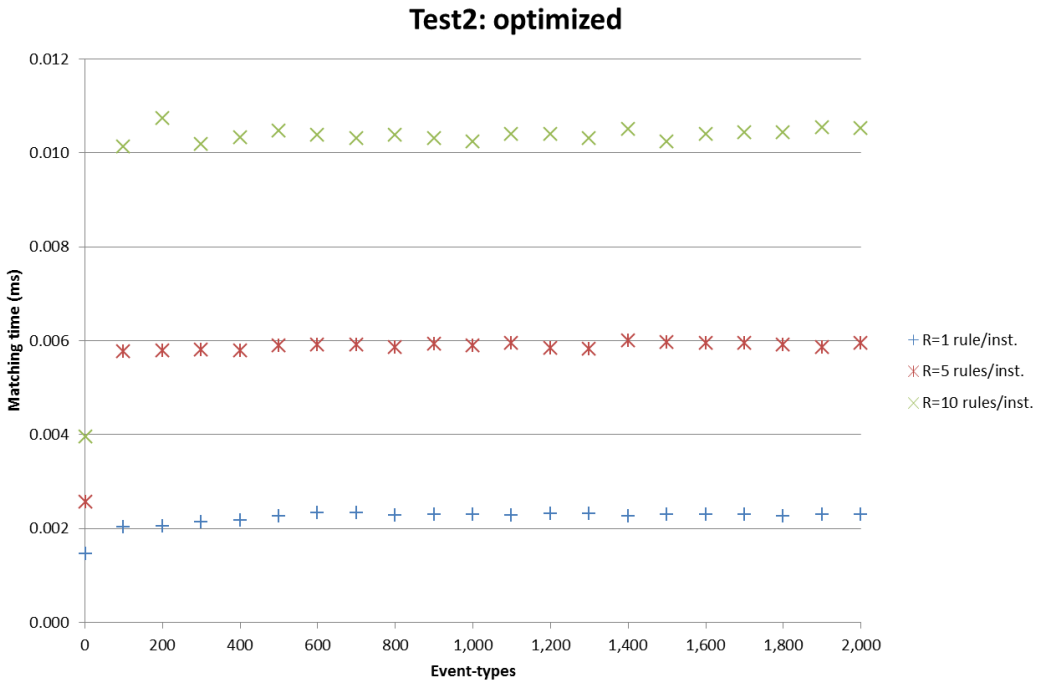


Figure 15. Matching time vs. event-types and rule repetitions under optimized conditions in Test 2.

5. Conclusions

This paper presents a meta-model for the definition of rule-based reactive behavior applicable to the development of customized smart environments. The proposed model is a step forward with respect to existing state of the art rule-based engines in several ways: by supporting the definition of arbitrary complex data transformations on any information (event attributes and entities properties) available in the ecosystem so that complex behaviors may be implemented in the consequent section of any rule, not being confined to state actions; by supporting collections of entities both as sources and targets of any reactive rule in the form of bound variables, allowing the definition of rules triggered by behaviors associated to taxonomies of entities and that act simultaneously by updating properties or invoking services on collections of entities simultaneously; by supporting the definition of data transformation processes by means of a visual language based on data flows that can be effectively used by non-expert programmer fifth-year secondary school students.

Although a rule editor prototype for our model has already been implemented and functionally tested in a touch screen [29], an immediate future work is to improve it by including wizards, recommenders and auto-completion techniques to facilitate the guidance and scaffolding of the rule construction process. These improvements would facilitate the use of some specific rule elements that are more difficult for people with low skills such as bound variables and non-state actions corresponding to mathematical expressions. After that, a more ambitious future work is to deploy the prototype in an actual intelligent environment by implementing the binding with a specific AmI middleware and conduct then an empirical user based evaluation with actual inhabitants to obtain empirical evidence on which level of personalization users are able to achieve.

In addition, the expressiveness of the model requires some mathematical knowledge, and this means that it is not suitable for all kind of users. However, new user interfaces must be explored and devised, even at the cost of limiting the full language expressiveness, to offer several editing tools suitable to users with different skills.

The tools implemented until now used to validate the model are currently being redesigned. Specially, the rule editor prototype is being re-implemented on a tangible surface interface to make environments with this type of technology more natural and seamless, or even could bring the chance to explore how rules can be edited collaboratively. As the performance verification conducted in this paper focuses on a very specific issue but showed the potential of using the meta-model, several aspects are interesting for further development. For example, in the future we plan to explore rule enactment distribution techniques, so that the control and coordination processes can be shared among several computing nodes to cope with highly populated ecosystems with even complex rules or other constraints related to low-end computing nodes. In addition, the refinement of mechanisms to deal with conflicts is also interesting, as well as using a more complex priority-based scheme to rank rule instantiations in those conflicting situations.

Acknowledgment

This work received financial support from the Spanish Ministry of Education under the National Strategic Program of Research and Project TSI2010-20488. Our thanks also to the high school "Col•legi Parroquial D. José Lluch – Alboraya", especially to the teachers and students that participated in the empirical study reported in this paper. A. Catala is supported by a FPU fellowship from the Ministry of Education of Spain with reference AP2006-00181.

References

[1] N. Shadbolt. Ambient Intelligence. IEEE Intelligent Systems 18, 4 (July 2003) 2-3.

[2] P. Remagnino, G.L. Foresti. Ambient Intelligence: A New Multidisciplinary Paradigm, Systems, IEEE Transactions on Man and Cybernetics, Part A: Systems and Humans, 35:1, (Jan. 2005), 1- 6.

[3] D. A. Norman, The Invisible Computer. Cambridge, MA: MIT Press, 1999.

[4] M. Weiser, The computer for the twenty-first century. Sci. Amer., 265:3, (1991), 94–104.

[5] J. Bravo, R. Hervás, I. Sánchez, G. Chavira, S. W. Nava. Visualization Services in a Conference Context: An Approach by RFID Technology, J. UCS 12:3 (2006), 270-283.

[6] A. Uribarren, J. Parra, J. P. Uribe, M. Zamalloa, K. Makibar. Middleware for distributed services and mobile applications. In Proc. of InterSense '06. ACM, 2006.

[7] D. López-de-Ipiña, J. I. Vázquez, D. García, J. Fernández, I. García, D. Sainz, A. Almeida. EMI2lets: A Reflective Framework for Enabling AmI. J. UCS 12:3 (2006), 297-314.

[8] C. Becker, M. Handte, G. Schiele, K. Rothermel. PCOM - a component system for pervasive computing. In proc. of PerCom'04, pp. 67- 76, 2004.

[9] L. Fuentes, D. Jiménez, M. Pinto. Development of Ambient Intelligence Applications using Components and Aspects. J. UCS 12:3 (2006), 236-251.

[10] N. Gámez, L. Fuentes. FamiWare: a family of event-based middleware for ambient intelligence. Personal and Ubiquitous Computing, 15:4 (2011), 329-339.

[11] J. B.Lézoray, M.T. Segarra, A. Phung-Khac, A. Thépaut, J.M. Gilliot, A. Beugnard. A design process enabling adaptation in pervasive heterogeneous contexts. Personal Ubiquitous Comput. 15:4 (April 2011), 353-363.

[12] A. Uribarren, J. Parra, R. Iglesias, J. P. Uribe, D. López-de-Ipiña. A Middleware Platform for Application Configuration, Adaptation and Interoperability. In Proc. of SASOW '08, pp. 162-167, 2008.

[13] T. v. Kasteren, A. Noulas, G. Englebienne, and B. Kröse. Accurate activity recognition in a home setting. In Proc. of UbiComp '08. ACM, pp. 1-9, 2008.

[14] Y. Uhm, M. Lee, Z. Hwang, Y. Kim, S. Park, A multi-resolution agent for service-oriented situations in ubiquitous domains, Expert Systems with Applications, 38:10, (15 September 2011), 13291-13300.

[15] J. Yang, J. Lee, J. Choi. Activity recognition based on RFID object usage for smart mobile devices. J. Comput. Sci. Technol. 26, 2 (March 2011), 239-246.

[16] J. Jaen, J. A. Mocholí, A. Catala, E. Navarro. Digital ants as the best cicerones for museum visitors, Applied Soft Computing, 11:1, (January 2011), 111-119.

[17] A. Picón, S. Rodríguez-Vaamonde, J. Jaén, J. A. Mocholi, D. García, A. Cadenas. A statistical recommendation model of mobile services based on contextual evidences, Expert Systems with Applications, 39: 1, (January 2012), 647-653.

[18] M. Rodríguez, D. M. Escalante, A. Peregrín. Efficient Distributed Genetic Algorithm for Rule extraction, Applied Soft Computing, 11:1, (January 2011), 733-743.

[19] Yu-Wang Chen, Jian-Bo Yang, Dong-Ling Xu, Zhi-Jie Zhou, Da-Wei Tang. Inference analysis and adaptive training for belief rule based systems, Expert Systems with Applications, Volume 38, Issue 10, 15 September 2011, Pages 12845-12860.

[20] T. Weis, M. Handte, M. Knoll, C. Becker. Customizable Pervasive Applications. In Proc. of PerCom'06, pp. 239-244, 2006.

[21] L. Fuentes, N. Gámez. Configuration Process of a Software Product Line for AmI Middleware. J. UCS 16(12): 1592-1611 (2010).

[22] T. Weis, M. Knoll, A. Ulbrich, G. Muhl. A. Brandle. Rapid Prototyping for Pervasive Applications. Pervasive Computing, IEEE, 6:2 (April-June 2007), 76-84.

[23] P. Giner, C. Cetina, J. Fons, V. Pelechano. A Framework for the Reconfiguration of Ubicomp Systems. 3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008, Advances in Soft Computing, Springer Berlin / Heidelberg, pp. 1-10, vol. 51, 2009.

[24] T. Sohn, A.K. Dey. iCAP: An Informal Tool for Interactive Prototyping of Context-Aware Applications. Extended Abstracts of CHI 2003, pp. 974-5, 2003.

[25] Y. Li, J. I. Hong,, J. A. Landay. Topiary: a tool for prototyping location-enhanced applications. In Proc. of UIST '04. ACM, pp. 217-226, 2004.

[26] T.Zhang, B.Bruegge, Empowering the User to Build Smart Home Applications. Second International Conference On Smart homes and health Telematics.Singapore, August 15-17, 2004.

[27] C. Beckmann, A. K. Dey. SiteView: Tangibly programming active environments with predictive visualization. Technical Report IRB-TR-03-019, Intel Research Berkeley, July 8, 2003.

[28] J. M. Wing,. Computational thinking. Commun. ACM vol.49, no.3, pp.33-35, March 2006.

[29] P. Pons, A. Catala, J. Jaen, J. Mocholi. DafRule: Un modelo de Reglas Enriquecido mediante Flujos de Datos para la Definición Visual de Comportamiento Reactivo de Entidades Virtuales, Actas de las Jornadas de Ingeniería del Software y Bases de Datos, "JISBD 2011", pp. 989-1002, 2011.

[30] A. K. Dey, R. Hamid, C. Beckmann, I. Li, and Daniel Hsu. a CAPpella: programming by demonstration of context-aware applications. In Proc. of CHI '04, pp. 33-40, 2004.

[31] M. García-Herranz, P. A. Haya X. Alamán. Towards a Ubiquitous End-User Programming System for Smart Spaces. J. UCS 16:12, (2010), 1633-1649.

[32] D. Bonino, F. Corno, L. De Russis. A User-Friendly Interface for Rules Composition in Intelligent Environments., In Proc. of ISAMI2011, Advances in Intelligent and Soft Computing Series, Springer Berlin / Heidelberg, pp. 213-217, vol. 92, 2011

[33] C. Kelleher, R. Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Comput. Surv. 37, 2, pp. 83—137, June 2005.

[34] J.F. Pane, C.A. Ratanamahatana, B.A. Myers. Studying the Language and Structure in Non-Programmers Solutions to Programming Problems, International Journal of Human-Computer Studies, 54:2 (February 2001), 237-264.

[35] J. Good, K. Howland, K. Nicholson. Young People's Descriptions of Computational Rules in Role-Playing Games: An Empirical Study, In Proc. of VL/HCC 2010, pp. 67-74, 2010.

[36] D. López-de-Ipiña. An ECA Rule-Matching Service for Simpler Development of Reactive Applications, In Proc. of Middleware 2001, IEEE Distributed Systems Online, 2: 7 (November 2001).

[37] I. Kurtev, J. Bézivin, M. Aksit.Technological Spaces: An Initial Appraisal. Int. Federated Conf. (DOA, ODBASE, CoopIS), Industrial track, Irvine, 2002.

[38] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev. 2008. ATL: A model transformation tool. Sci. Comput. Program. 72:1 (June 2008), 31-39.

[39] R. Hervás, J. Bravo, J. Fontecha, A Context Model based on Ontological Languages: a Proposal for Information Visualization, J. UCS 16:12 (2010), 1539-1555.

[40] K. N. Whitley, L. R. Novick, D. Fisher. Evidence in favor of visual representation for the dataflow paradigm: An experiment testing LabVIEW's comprehensibility. *Int. J. Hum.-Comput. Stud.* 64:4 (2006), 281-303.

[41] L. W. Anderson, D. R. Krathwohl, P. W. Airasian, K. A. Cruikshank, R. E. Mayer, P. R. Pintrich, J. Raths, M. C. Wittrock, A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon (2000).