

Dealing with software process deviations using fuzzy logic based monitoring

Sorana Cîmpan, Flavio Oquendo

Abstract-- The work presented in this paper concerns the monitoring of software processes using a fuzzy logic based approach. The proposed monitoring for software processes focuses on the detection of deviations with respect to given models. The level of deviation is computed for different aspects of the process like progress, cost, structure (order between activities), etc. The proposed approach is implemented by the OMEGA environment (OMEGA stands for On-line Monitoring Environment: General and Adaptable). The environment provides a language for defining monitoring models. Such models are executed by the environment's engine. Fuzzy sets theory is used in the representation of the information handled by monitoring systems, and possibility theory in the reasoning upon it. Fuzzy logic offers significant advantages over other approaches due to its ability to naturally represent uncertain and imprecise information. It equally provides a good framework for approximate reasoning. The use of the proposed monitoring environment is illustrated in the context of the Unified Software Development Process.

Index Terms--Monitoring, software processes, fuzzy logic.

I. INTRODUCTION

The success of software engineering organizations heavily depends on the quality of their software products which is intrinsically related to the quality of the software development processes. Technology for supporting organizations taking a process-oriented approach has been developed during the past decades, having as ultimate goal to reach the point where the process representation (process model) can be used throughout its enactment in order to drive the actual process of software development (process performance). A software process can thus be structured in three domains [14]: *process definition*, *process performance* and *process enactment*. These domains as well as the interactions between them are illustrated in Figure 1.

A process model is instantiated for a given project, information specific to the project being added. The instantiated process model is an enactable representation of the process. Its enactment by an enactment engine gives the enacting process. The enactment is not deterministic, and takes into account the feedback from process performance.

The leak of process fitness for a particular project may lead to situations where the process model is not faithfully followed, so where process fidelity [19] is absent. The performed process may become in such situations non-conformable with respect to the instantiated process model, i.e. starting from the later no process enactment conformable with the performed process can be created. Such non-conformance, or deviations, may also be caused by changes in the context in which the process takes

place. The observed deviations may or may not lead to evolution of the process model. Regular deviations may be the sign of a process that was not well enacted, not well instantiated or further more, not well modeled, and thus subject for improvement.

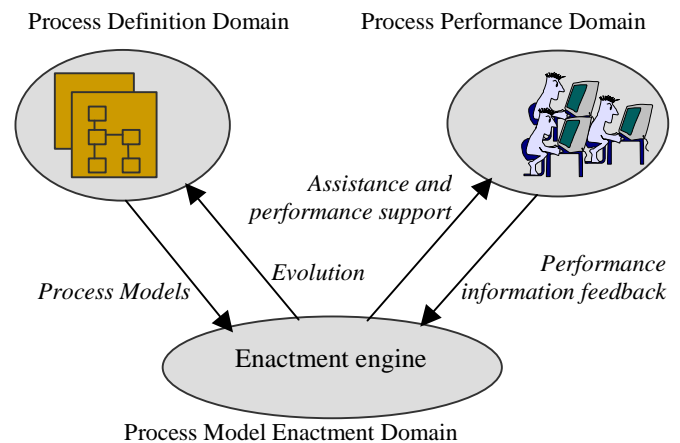


Figure 1. Software Process Domains

Let us assume that the performed process deviates with respect to its associated model. If the process support environment allows deviations between the (instantiated) process model and the enacting process, the later can remain conformable with the performed process. The deviations between the performed process and the process model become deviations between the process model and the enacting process. Such deviations as well as other aspects of the process (including the products being produced) can be observed in an automated manner using a monitoring system. The later can measure and keep under control in an automated manner the level of conformance between the process model (instantiation) and the performed process. A restricted representation of the performed process can also be constructed at the level of the monitoring system. Such representation concerns the attributes of the process the manager is interested in monitoring. The existence of a monitoring system in the process support environment follows the requirements stated in the most advanced levels of Software Engineering Institute CMM process maturity framework [22], “managed” and “optimizing” processes (levels 4 and 5), where the use of monitoring techniques is required and the use of these systems is highlighted.

We propose in this paper an environment for monitoring software processes: the *OMEGA* environment (OMEGA stands for

On-line Monitoring Environment: General and Adaptable) [7][8]. The environment provides the language *OMEGA/MDL* (*Monitoring Definition Language*) for defining monitoring models as well as a mechanism for the execution of such models *OMEGA/EM* (*Execution Mechanism*). The executing monitoring models (i.e. monitoring systems), observe the process and detect deviations between it and an expected behavior, i.e. indicated by the process model (instantiation).

OMEGA proposes a novel approach for monitoring [10][7], based on fuzzy logic [26]. This approach allows to establish the level of conformance between the performed process and the process model (instantiation) for different aspects of the process, like progress, cost, structure (order between activities), etc. The level of conformance varies from *total conformance* to *no conformance at all*. The monitoring system is integrated in a quantitative control system which assists the process manager in the control and evolution of the process s/he is in charge of: it detects deviations and supports the reconciliation of the ones that are unacceptable by indicating possible corrective actions and implementing them [1][2]. The use of fuzzy logic enables the system to handle imprecise and uncertain as well as precise and certain information. Thus, the fuzzy sets theory is used in the information representation, while the possibility theory [15] is used in the reasoning upon imprecise information (with the help of fuzzy rules). OMEGA is also open to evolution and allows the integration of new data collection and analysis techniques. OMEGA was conceived to work in a process support environment, where the collection of information can be largely automated (as the information is collected from the process model enactment domain) and a representation of the process model is generally available. Nevertheless, OMEGA can be used for monitoring software process in the absence of such a process support environment. On the one hand, the information collection can be made from other sources of information (like process performers, databases, etc.). On the other hand, a monitoring model generally focuses on some particular aspects of the process, and the expected behaviour of the process with respect to these aspects can be modeled inside the monitoring model (eventually starting from process trace databases). In this paper we will illustrate the use of OMEGA in the context of the Unified Software Development Process [24], throughout a monitoring case study, in which the existence of a process support is not mandatory.

The paper is further structured as follows: in the following section we present the software process monitoring approach adopted in order to handle deviations. Section III presents its implementation by the OMEGA environment. Section IV presents a monitoring example and illustrates its use in the context of the Unified Software Process. Section V is dedicated to an overview of the related work.

II. PROCESS MONITORING: AN OVERVIEW OF THE APPROACH

We see the process monitoring as a control technique entailing measurements and detection of deviations with respect to an expected behavior. The way the detection of deviations is made is by computing for a process the level of conformance to an expect behavior: the lower the conformance, the higher the deviation. The conformance is a measure of similarity, and expresses to which extent the observed process behaves similarly to what is expected (possibly indicated by the process model).

When a process model is defined and instantiated for a given project, the expected behavior is represented by the enactment of the process instantiation, in a context where no unexpected events occur. The process instance is completely conformable to the instantiated process model. Deadlines are respected, every single product is produced in time and in budget, with the resources allocated at process model instantiation. One might argue that this is not an expected behavior, but an ideal one, and that is fair. When launching a new project, people not always expect such an ideal behavior. Some deviations are expected. The issue is then what are the accepted deviation? How much can one deviate from the ideal behavior without having major impacts on the process outcome? As it will be further shown, the monitoring approach allows to model the expected behavior in flexible manner, indicating the ideal behavior as well as the tolerated deviations.

The conformance is not considered globally, but for individual aspects¹. For each considered aspect a conformance factor is considered, given a more detailed view of the process and its conformance to the expected behavior. We have constructed conformance factors related to the structure of the process (“are the dependencies between activities respected?”), to the advancement of activities (“will the activity be completed within the established deadlines?”), etc. The later will be presented in section IV.

For each process aspect to be monitored for conformance, a monitoring model is constructed and then enacted. The process behind the model construction and their enactment, called *P2M* (*Process for Process Monitoring*), is presented in the following subsection.

A. The Process for Process Monitoring (P2M)

Monitoring a software process is a process itself. We present this process throughout the three major phases of its life-cycle: definition, instantiation and enactment (see Figure 2).

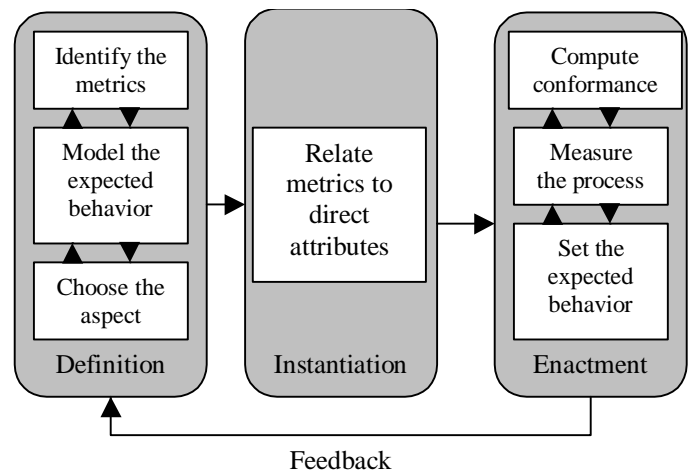


Figure 2 The Process for Process Monitoring

In the definition phase a monitoring model is constructed. In the instantiation phase, the model is related to a context of use. The enactment phase entails the actual monitoring of a given software process.

¹ An aspect can be associated to a process attribute

Feedback is provided from the enactment domain to the definition domain. This feedback is used in order to improve the monitoring models. This feedback can pass through meta-monitoring, i.e. through monitoring the monitoring enactment. The monitoring itself can be thus applied in a reflexive manner.

1) *The definition phase*

In the definition phase the monitoring model is constructed. This phase entails three main steps:

- Step 1: choose an aspect for which the conformance will be computed;
- Step 2: model the expected behavior;
- Step 3: identify the metrics (direct and indirect) which allow to compute the observed behavior for the chosen aspect.

Step 1. At step one the user decides what is the main aspect to be monitored for conformance. The aspect is generally associated to an indirect process attribute, i.e. the value of this attribute can be obtained from the value of other process attributes [20]. The direct attributes are the ones that are directly measurable on the process. The value of indirect attributes is computed out of the value of other attributes (direct or indirect). For instance the number of lines of code is a direct attribute, while the advancement of the coding activity (in terms of completeness vs. deadlines) is an indirect attribute.

Step 2. The first step is closely related to the second, in which the expected behavior is modeled. The modeling of the expected behavior consists in establishing:

- the aspect domain of representation;
- how the conformance should be computed;
- what are the associated thresholds and alarms.

As the aspect is in most of the cases an indirect attribute, its domain of representation has to be established. At this point one can decide whether to use linguistic or numeric representations, to accept or not imprecise or uncertain representation of the information, etc.

Equally to be established in this step is the way the conformance will be computed. In fact, ones the aspect domain chosen, two variables will be defined on this universe: the *tolerance* and the *observed aspect*. The tolerance variable (that takes values at the enactment time) indicates what are the ideal values as well as the accepted deviations concerning the aspect for which the conformance is computed. The observed aspect will take, at enactment time, the value for the considered aspect as computed using the metrics identified in the step 3. The conformance is the result of the comparison between the tolerance and the observed aspect. As expected, the used formula depends heavily on the considered aspect domain.

Associated to the conformance are the *thresholds* and the *alarms*. When the threshold value is crossed, an alarm is raised by the monitoring system. Generally the threshold is represented in the same domain (numeric or linguistic) as the conformance in order to ease the comparison between the two.

Step 3. The third step corresponds to the identification of the metrics to be used in order to compute value for the observed aspect, i.e. how starting from direct process attributes one can obtain the value of the observed aspect. If no such metrics can be identified, the process loops to the previous steps.

One can notice that we have taken an iterative approach in the definition of monitoring models. In fact, as shown in Figure 2, one can return from step 2 to step 1 (difficulty to correctly model the expected behavior for the chosen aspect, e.g. no appropriate

domain of representation is found, or conformance computing formula) or from step 3 to step 2 (no available metrics to capture the required information, e.g. difficulties in finding the metrics that allow to calculate the value of the observed aspect out of the values of direct attributes).

The result of the definition phase is thus a monitoring model, which can be rather general. In order to be used in the context of a given software process, it has to pass through the instantiation phase.

2) *The instantiation phase*

The instantiation phase has the purpose to put the monitoring model in a context of use. The instantiation is made at two levels:

- for a given software process: relates part of the metrics to direct attributes of the process;
- for a given software project that uses the software process: indicates what are the sources for the direct measures, i.e. exactly where the information is to be searched (for instance indicating the name of a database), who are the persons that would provide the information, or what are the events to which the monitoring system has to subscribe in order to obtain the values of the attributes.

The instantiation for a given project is influenced by the existence of a process support environment. Generally, when such an environment exists, information on the software process is available in this environment, and the monitoring system can collect (possibly via an event publication/subscription mechanism) the values for most of direct attributes needed. In the same phase interfaces for manual collection of information might also be provided, for attributes which values are not available in the process environment.

Once this phase completed, the monitoring model is adapted for use with a given software project, and is ready to be enacted.

3) *The enactment phase*

An instantiated monitoring model is ready to be enacted. Its enactment by the monitoring engine represents a monitoring system. At the enactment phase the monitoring system interacts with its environment. On the one hand, the process is measured, on the other hand the results of the conformance computing are published.

The enactment phase cycles around the following steps:

- set the expected behavior: the values for tolerance and thresholds are established;
- measure the process: the system collects the values of direct attributes and computes those of indirect attributes (the value of the observed aspect is computed);
- compute conformance: the observed aspect is compared with the tolerance for detecting the conformance level;
- detect if unacceptable deviation appear: the conformance value is compared with its associated threshold; alarms are raised if the thresholds are crossed.

P2M was presented here in its form related to the detection of deviations. Lighter versions might be considered, for simpler monitoring system. For instance one may envisage a very basic monitoring system that only collects information and displays it or stores it in a trace database. In such a case the definition is simpler, as only direct attributes are to be identified, no conformance, alarms or thresholds are to be defined.

We had an overview of P2M, the process for process monitoring. In the following section we present the theoretical tool

underlying the construction of monitoring models: the fuzzy logic.

B. The use of fuzzy logic

As seen in the previous section, a monitoring system handles information. It collects it from the monitored process and uses it order to produce other information, concerning the level of conformance for instance. In order to handle this amount of information, a tool is needed to represent it and to reason upon it. We have chosen the fuzzy logic as underlying theory, namely the fuzzy sets theory is used in the representation of information, and the possibility theory in the reasoning upon information. General arithmetical operators are equally used for handling numerical fuzzy sets. Fuzzy logic offers significant advantages over other approaches due to its ability to naturally represent qualitative aspect of data and inference rules used.

The fuzzy sets theory was developed by Zadeh in order to represent mathematically the imprecision related to certain object classes [26]. It also responds to the need for representing symbolic knowledge, in natural language, submitted to imprecision or presenting a vague character. A fuzzy subset A of a set X is characterized by an application from X to $[0, 1]$. This application called membership function and noted μ_A allows to represent how much the elements x of X are also members of A . If $\mu_A(x)=1$ the value x is completely a member of A and if $\mu_A(x)=0$, x is not at all a member of A . In the particular case when $\mu_A(x)$ only takes values equal to 0 or 1, the fuzzy subset A is a classic subset of X .

Let's take the example of the completeness of an activity. It can be represented numerically in terms of achieved percentage (corresponding for instance to the advancement of the product that the activity has to produce), in the universe $[0, 100]$. Assume that the value of the attribute has to be provided by an actor involved in the considered activity. For some particular activities, the completeness can be provided in a precise and certain manner, and thus the classical set theory can be used in representing such information. In some other cases, such a crisp value cannot be provided, only imprecise or uncertain value being available. There are three possibilities to be considered:

- consider the information as unavailable;
- ask the user for a crisp value, and do not take into account the imprecise nature of the information;
- use a formalism that allows representing imprecise and uncertain information, taking into account their nature.

By using the fuzzy sets theory, we have chosen the third option. This allows the monitoring system to function even when only imprecise information is available.

The use of fuzzy sets theory gives the possibility to handle symbolic information. Considering the example of the completeness, it can also be represented in a linguistic universe that contains terms like $\{very\ low, low, medium, high, very\ high\}$. The completeness can thus take values in this universe, represented as fuzzy subsets. A value like $\{0/very\ low, 1/low, 0/medium, 0/high, 0/very\ high\}$ indicates that the completeness is low^2 , whilst a value like $\{0.8/very\ low, 0.2/low, 0/medium,$

$0/high, 0/very\ high\}$ indicates that the completeness might be ranked as *very low* or *low*. The actor is though more confident in ranking it as *very low* than *low*.

Figure 3 illustrates the representation of the completeness attribute in the linguistic universe above mentioned and the correspondence with the numerical universe $[0, 100]$. The fact that the system accepts uncertain and imprecise information (and represents such information), induced the need for a theory that allows to reason upon such information. Possibility theory [15] is used in this context. This choice was also motivated by the fact that a tool for approximate reasoning was needed in order to capture to more flexibly the domain knowledge concerning the different computations upon acquired information.

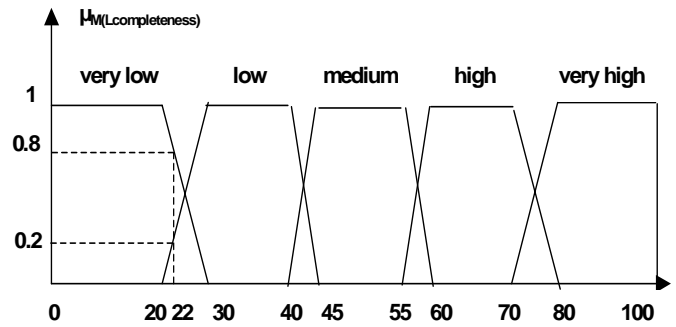


Figure 3 Correspondence between a numerical and a linguistic representation for the completeness attribute

The value of an indirect attribute can be obtained out of the values of other attributes (direct or indirect) using fuzzy rules. Let us consider the case where the value of an attribute Y has to be computed out of the values of attributes X_i ($i=1, n$). The fuzzy rules used are of the form:

$$\text{If } A_i(X_i \text{ is } A_i) \text{ then } Y \text{ is } B$$

A_i represent linguistic terms from the domains of representation of the attributes X_i ; A represents the conjunction operator. B is a linguistic term from the domain of representation of the attribute Y .

Starting from the membership degrees of X_i to different linguistic terms of their universes, the membership degree of Y to B is computed by intersection, where the product operator has been chosen as t-norm [14]:

$$\mu_B(Y) = \prod_{i=1, n} \mu_{A_i}(X_i).$$

If there are m rules with the conclusion Y is B then the degree of membership for Y to B is the computed from the respective j computed $\mu_{jB}(Y)$ using the Lukesiewicz t-conorm [14]:

$$\mu_B(Y) = \min \{ \sum_{j=1, m} \mu_{jB}(Y), 1 \}.$$

Due to its ability to naturally model the imprecise and uncertain aspect of data and rules, fuzzy logic is an attractive alternative in a situation where approximate reasoning is needed. A prototype system can also be developed based solely on domain knowledge without relying on extensive training data. Furthermore, performance of the system can be gradually tuned as more data become available.

² The strict interpretation is: the activity considered is a typical element of the set of activities considered to have a *low* completeness. The membership degree to this set is 1, while the membership degree to all

III. THE OMEGA ENVIRONMENT

OMEGA stands for *On-line Monitoring Environment: General and Adaptable*. It was conceived as an implementation of the approach proposed in the previous section. The environment is composed by:

- a *monitoring definition language*: OMEGA/MDL, that allows to define monitoring models;
- an *execution engine*: OMEGA/EM that executes monitoring models written in OMEGA/MDL.

The following of this section focuses on issues related to the definition of monitoring models in OMEGA. Another section is dedicated to OMEGA's architecture.

A. Defining monitoring models

A monitoring model indicates the behavior of the monitoring system resulted from its execution in terms of *analysis* and *communication* with the environment in which the system will be integrated. These concerns are separated in the OMEGA monitoring models, different specific sub-models being used for each of these aspects (we will refer to the monitoring sub-models as models or specific models). The sub-models are classified into analysis models and communication models.

An *analysis model* defines data to be collected from the subject process as well as analyses that are to be applied on the data. The data represents process attributes. OMEGA/MDL uses the fuzzy sets theory in representing the information.

Communication models are used to specify how the monitoring system communicates with the environment in which it is integrated. There are models for handling the inputs and models for handling the outputs of the monitoring system.

A *sensor model* is concerned with the input handling and indicates how the subject process has to be instrumented, i.e. which are the direct attributes that have to be measured and wherefrom their values are to be collected.

A *publication model* indicates the information that has to be published for the rest of the environment. Special outputs handling are considered for interfaces (in *display models*) and for trace databases (in *trace models*).

Independently of their special nature, the monitoring models contain the elements described below.

- *Import statements*: allow the access to definitions from other monitoring models.
- *Universe definitions*: allow extending the set of predefined universes in which the data to be collected or produced by the monitoring system is represented.
- *Information definition*: declare the data handled by the monitoring system. The information refer to direct or indirect attributes [20].
- *Transformations definitions*: declare the analysis to be applied to the information. They are used in order to obtain the values of indirect attributes.

The information as well as the transformations may have special roles, as it will be further shown.

OMEGA provides a set of operators to be used in the transformations. They allow the treatment of imprecise information (numeric or linguistic) using fuzzy arithmetic, fuzzy rules sets, etc.

More details on the language are presented in the section IV where the introduced concepts are illustrated by showing how a monitoring case study is modeled using OMEGA/MDL.

B. OMEGA's architecture

As mentioned earlier, a monitoring model is composed by different models with specific roles (analysis or communication). These models are instantiated and then executed by OMEGA/EM. The models are compiled and java classes are generated. The java classes are in their turn compiled into binary files, which together with the binary files of the execution mechanism compose a specific monitoring system.

When a new value for an information is captured, all the transformations that have the given information in the input are triggered, and their effect (changes other the information values) is afterwards propagated. The publication transformations are applied to publish (display or trace) the changed information. Figure 4 presents OMEGA's architecture and the way it is integrated in a process support environment (indications about implementation choices are also provided). The monitoring is also structured according to the three software process domains. Thus, for monitoring there is a definition domain (where the monitoring models are created), an enactment domain (where the monitoring models are executed), and a performance domain (where the monitoring is performed). The process manager acts in the later.

The data collection is made from the subject process enactment and performance, as well as from the monitoring performance domain. A middleware is used for the communication with the subject process enactment support, which allows the collection of information throughout events.

Other components may be provided, completing the feedback loop (such as decision support, change support, etc.). When this is the case, the outputs of the monitoring may be redirected towards such components. The communication between the monitoring enactment and the subject process enactment is made by a subscription/publication mechanism.

IV. AN ILLUSTRATION OF THE APPROACH

In the previous sections we have seen an approach for monitoring software processes and its implementation in the OMEGA environment. A monitoring case study (presented in the section A) illustrates the kind of monitoring one can envisage with OMEGA. The corresponding monitoring model in OMEGA/MDL is presented in the section B, followed by an example of usage in the Unified Software Process (section C).

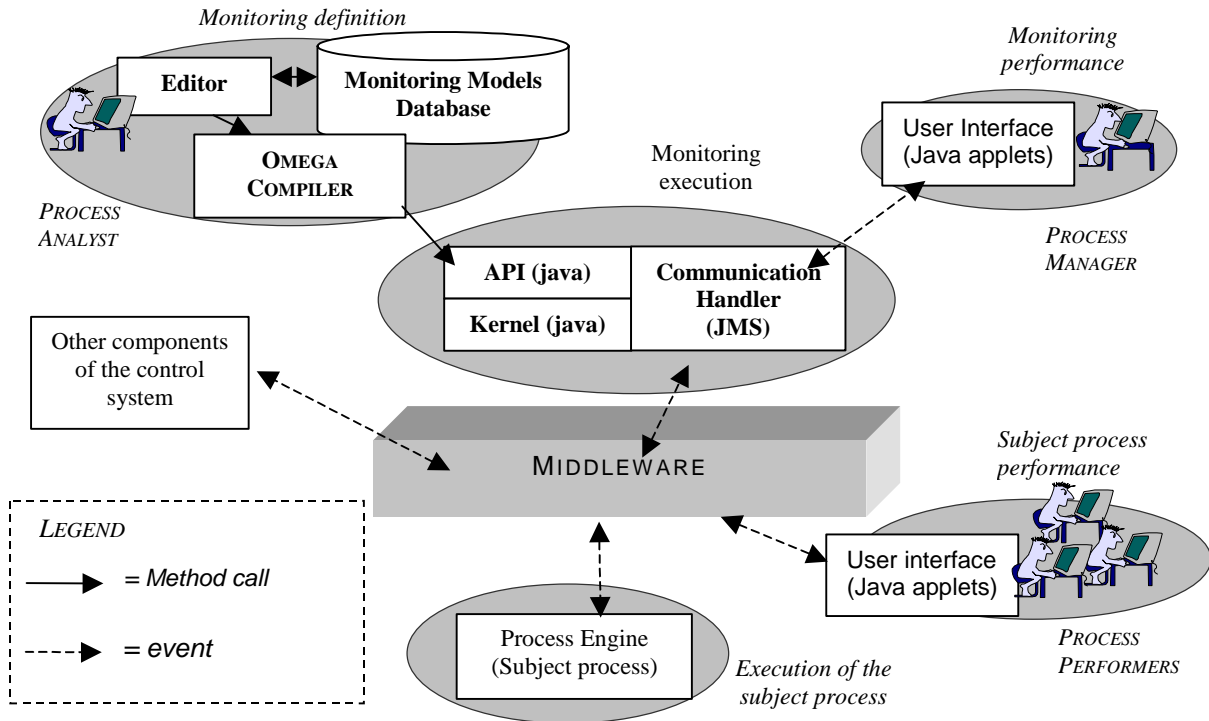


Figure 4 Omega's architecture

A. *Monitoring case study: The confidence in deadlines preservation*

The considered case study concerns the monitoring of process activities in order to establish the level of confidence in that their completeness will be achieved by the moment they are supposed to finish, i.e. that the deadlines will be preserved. Starting from collected process data concerning the completeness of the work to be achieved and the position in time with respect to deadlines, the above mentioned confidence is computed. This information is useful as the activity progresses (especially when keeping the deadlines is crucial) and gives the possibility to react with corrective actions whenever the confidence decreases. We describe hereafter the various informations handled, as well as the way the confidence is computed out of the collected data. The *completeness* is represented in terms of achieved percentage. The deadlines are represented throughout three values:

- T_A : the time allocated for the activity. It corresponds to the time estimated for completing the activity without taking risks, as it includes reaction time (i.e. time for doing some rework in case of error detection or unexpected situations).
- T_M : the minimal time to reach the activity completeness. It leaves no reaction time at all, and any minor problem induces the incapacity of reaching the activity completeness in the established deadlines.
- T_I : the intermediary time, between the allocated and the minimal ones. It allows completing the activity without leaving much reaction time. Some risks are associated with it.

The values $T_A - T_I$ and $T_A - T_M$ can be associated, at 0% completeness, to alert and alarm thresholds. For other completeness values alert and alarm thresholds are generated during the enactment, as it will be explained further in the paper.

Figure 5 presents these time values in a space where one axe represents the time, and the other the completeness. In this space some zones are identified³.

The confidence is represented in a linguistic universe containing the values {*very high, high, normal, low, very low*}, as a fuzzy subset of this universe.

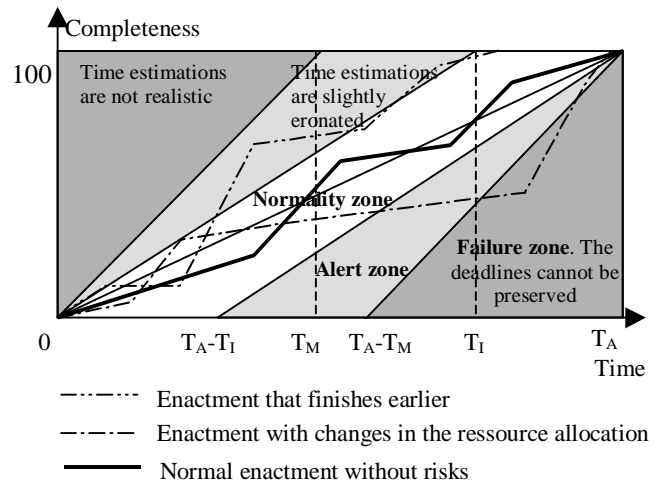


Figure 5. The completeness-time space and the associated risk zones

In the case of fuzzy linguistic subsets, for each linguistic term of the universe, the membership degree⁴ is indicated. For the

³ The model is given for illustrative purpose. Some simplifications were made in order to increase readability (for instance the linear approach in separating the different zones). Further refinements are possible in order to adapt it for taking other aspects into account.

⁴ The membership degree is the value of the membership function.

confidence of achieving the completeness within the established deadlines, an example of value is $confidenceExample = \{0/very\ high, 0/high, 0/normal, 0.7/low, 0.3/very\ low\}$: the confidence is evaluated as being between *low* and *very low*

$$(\mu_{low}(confidenceExample)=0.7, \mu_{verylow}(confidenceExample)=0.3).$$

Relations can be established between numeric and linguistic universes. Thus, if for a given linguistic universe there is an associated numeric universe, each linguistic term has associated a fuzzy subset of the numerical universe and vice-versa (to each numerical value a fuzzy linguistic subset is associated). In our example a relation is established between the confidence linguistic universe and the time universe.

The membership functions of the fuzzy numerical subsets associated to each term are dynamically generated by the monitoring system for each completeness value (see Figure 6). Let us take the example of the term *low*. The associated fuzzy number is related to the alert threshold. A function gives the value of this threshold depending on the completeness (in our case this function is linear, for the completeness 0% returns $T_A - T_I$ and for completeness 100% returns T_A). The alert threshold represents a typical element of the fuzzy number associated to the term *low*.

The monitoring system computes the confidence values as the time passes (every considered time unit) and the completeness of activity changes.

For each completeness value, and taking into account the three above mentioned deadline values, the monitoring system generates the thresholds and the membership functions associated to each confidence value (see Figure 6).

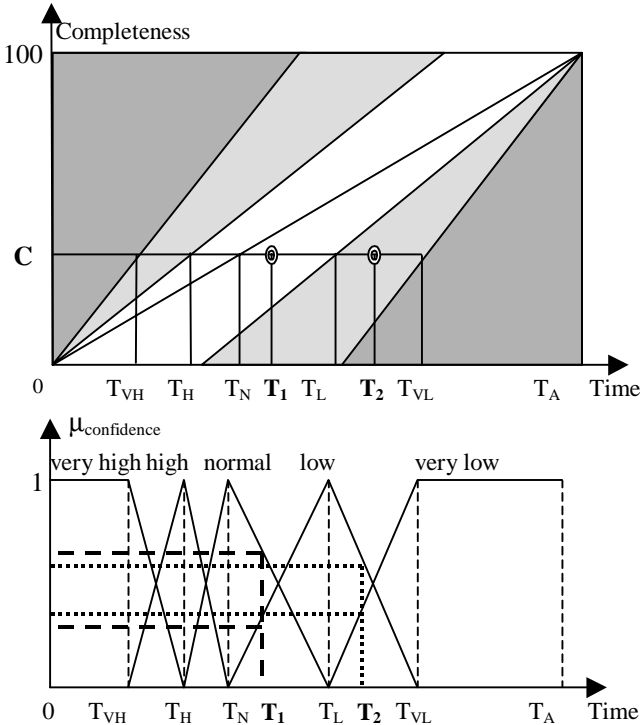


Figure 6. Generated membership function for a given completeness value

In Figure 6 T_{VL} is associated to the alarm threshold at C completeness and corresponds to the moment that is totally representative for the confidence value *very low*, T_L is associated

to the alert threshold at C completeness and corresponds to the moment that is completely representative for the confidence value *low*, etc. An alarm is used for signaling the entrance in the alert zone or in the failure zone. The alarm is defined on the universe $\{no-problems, alert, alarm\}$ and represents a fuzzy subset of this universe. The values of this alarm can be directly associated with the confidence value, by associating *alert* with *low* confidence and *alarm* with *very low* confidence. For instance when the computed confidence is *low*, the alarm is set to the value *alert*. More precisely, a set of fuzzy rules makes then the connection between the two universes. For the given example, the membership degree of the alarm to the *alert* value is the same as the membership degree of the observed confidence to the term *low*.

However, this correspondence can be defined in a more flexible manner using tolerance values for the observed confidence. The tolerance contains the target values as well the acceptable deviations. It is established by the process manager and can be dynamically changed during the enactment. The manager can decide for instance that the tolerated values are *high*, *normal* and *low*, each of them to a certain degree. In another context it might be considered that the value *low* is not tolerated. The observed confidence is compared with its associated tolerance and the result of this comparison gives the conformance of the enactment to the tolerated values (for the confidence of reaching the completeness within delays). Alert and alarm thresholds are established for the conformance value. The positioning of the alarm to different values is in this case resulted from the comparison of the conformance with the alert and alarm thresholds.

B. Defining the confidence model using OMEGA/MDL

An OMEGA model allows the definition of the monitoring system that implements the case study presented earlier (the model contains an analysis model and different communication models). We present in what follows the steps that lead to its construction using P2M. Extracts from this model are equally presented.

1) Choose the aspect

In this step it is decided that the aspect to be monitored for conformance is the reaching activity completeness within deadlines.

2) Model the expected behavior

At this step is decided that the confidence is represented in a fuzzy linguistic domain. The alarms associated to the conformance will also have the same type of representation.

```

universe Confidence = fuzzy { "very low", "low",
"normal", "high", "very high" };
universe Alarms = fuzzy {"inactive", "alert", "alarm"
};

```

The information defined in these universes can be imprecise or uncertain (represented as fuzzy subsets - keyword *fuzzy* in the universe definition).

Information is defined in order to represent the process attributes and other connected information (like thresholds and alarms). An information definition contains the keyword *information*. The language allows the definition of information having special roles like alarm, tolerance, conformance or threshold, using corresponding keywords in the beginning of their declaration. **information** observedCFD: Confidence ;/*the confidence in deadlines preservation*/

```

tolerance information toleranceCFD: Confidence;
conformance information conformanceCFD: [0,1];
threshold information alertThreshold: [0,1];
threshold information alarmThreshold: [0,1];
alarm information alarmeCFD : Alarms ;

```

The definitions above indicate the special role of information (thresholds, conformance, etc.). The way they relate to each other, i.e. the metrics used is indicated with the help of transformations.

3) Identify the metrics

Each information has one or more possible sources. From the modeling point of view, such sources are always transformations (metrics). When the value of the information is to be collected directly from the process, the source is a sensor transformation indicating which are the sensors that provide the information. When the information is an indirect attribute, its source is an analysis transformation. The sensors can be human or software, which correspond respectively to the fact that the information is provided explicitly by a process agent or can be capture in an automated manner (for instance under the form of an event raised from an active database).

The transformations contain a list of inputs and an output, these parameters corresponding to previously defined information. They also contain a statement part, which indicates the way the output is obtained from the input. OMEGA/MDL provides operators for relational, logical and arithmetical expressions, as well as operators for the treatment of uncertain, imprecise information (sets of rules, associations between numeric and linguistic universes, etc.).

Related to the confidence, we illustrate the transformation concept with the transformation that computes the value of the confidence (*observedCFD*). This transformation uses the *membershipFunctions* operator, which allows making the association between linguistic and numeric universes. The fuzzy numeric subsets are specified using the notation $[a\#b\#c]$ for the fuzzy numbers and $[a\#b\#c\#d]$ for the fuzzy intervals (a, b, c and d are real numbers). In Figure 6, the membership function associated to the term *low* is the fuzzy number $[T_N\#T_L\#T_{VL}]$, while for the term *very low* the corresponding membership function is given by the fuzzy interval $[T_L\#T_{VL}\#T_N\#T_N]$. The transformation is called *computeConfidence* and the fuzzy numbers and intervals associated with the linguistic terms are computed out of the values of the completeness (C) and the three time values linked to the deadlines (T_A, T_I and T_M).

```

transformation computeConfidence{
  input T,C, TA, TM, TI; /* T represents the time */
  output observedCFD;
  statement observedCFD= membershipFunctions (
    ("very high", [0#0#(C*TM/100)#(C*TI/100)]),
    ("high", [(C*TM/100)#(C*TI/100)#(C*TA/100)]),
    ("normal", [(C*TI/100)#(C*TA/100)#(TA-TI-
C*TI/100)]),
    ("low", [(C*TA/100)#(TA-TI+C*TI/100)#(TA-
TM+C*TM/100)]),
    ("very low", [(TA-TI-C*TI/100)#(TA-TM-
C*TM/100)#TA#TA]))}

```

The various formulas that appear in the fuzzy subsets definition correspond in Figure 6 to the generated T_{VL}, T_L , etc. For instance $T_{VH}=C*TM/100$ and $T_{VL}=TA-TM-C*TM/100$.

In the communication models (sensor, publication, display and trace models), the transformation definitions may be preceded by keywords that indicate their special nature. For example in the

sensor model the transformation that indicate the sensors for a given information are preceded by the keyword *sensor*.

The communication transformations are used to indicate how to collect the information (in the case of sensor transformation) or how to publish its values for the rest of the environment.

OMEGA's communication is completely event-based, the information collection and publication passes throughout events.

The sensor transformations are used to construct the event patterns for information collection. The publication transformations are used to construct event types associated to information to be published (events of these types will be raised whenever the information changes). In order to allow the event patterns and types construction, the communication transformation indicate in their statement part: the attribute name and type, the process and the entity names, the source and the destination of the information, etc. In the sensor transformations the statement part may contain only part of the elements mentioned, as they are used to construct the patterns (which may be rather general).

Hereafter we give a part of the sensor model used for the monitoring model described so far.

```

...
instantiation information sourceAgent: String;
instantiation information completenessName: String;

...
sensor transformation CollecteCompleteness {
  output C;
  statement human-sensor;
  attribute-name: completenessName;
  source: sourceAgent;
}

```

SourceAgent and *completenessName* are instantiation information, which means that their value is given when the model is instantiated, and is not changed during enactment. With its help information sources may be indicated. The transformation *CollecteCompleteness* is used for the collection of the completeness. It indicates that the sensor is a human sensor, and that the information C is encountered in the environment under the name *completeness*.

The monitoring system can collect the information regarding the completeness of the work from the process model enactment domain, or from the process performance domain, if a process support is not available providing the completeness values. The data related to the deadlines may come from the same domain (when represented in the process support), or can be provided from the monitoring performance by the project manager. The project manager during the enactment gives the confidence tolerance and conformance thresholds.

C. Monitoring instantiation and enactment for the Unified Software Development Process

After the definition phase, which we have just seen in the previous section, a monitoring model is instantiated and enacted for a given software project. We will illustrate these phases for the Unified Software Development Process [24], but before starting this illustration, we will make a small presentation of the considered process.

The Unified Software Development Process has been recently defined by Jacobson, Booch and Rumbaugh [24]. It is a generic process framework that can be specialized for a very large class of software systems. The Unified Process is a component-based

process that is use-case driven, architecture-centric, iterative and incremental.

The Unified Software Development Process repeats over a series of cycles making up the life of a system. Each cycle concludes with a product release to customers. Each cycle consists of four phases: inception, elaboration, construction and transition (see Figure 7). Each phase is further subdivided into iterations. In every iteration, the developers identify and specify relevant use cases, create a design using a chosen architecture as a guide, implement the design in components, and verify that the components satisfy the use cases. If an iteration meets its goals development proceeds with the next iteration. When an iteration does not meet its goals, the developers must revisit their previous decisions and try a new approach.

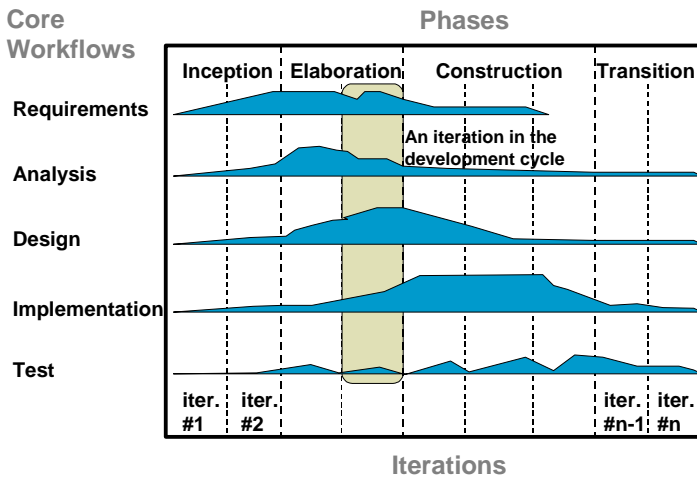


Figure 7 The five workflows over the four phases

The software product has different representations, provided by a set of models, containing: a use case model (*U*), an analysis model (*A*), a design model (*D*), an implementation model (*I*), a deployment model (*D*), a test model (*T*) and a representation of the architecture. All these models are related and together they represent the system as a whole.

Each phase of the lifecycle terminates in a milestone. The later is defined by the availability of a set of artifacts; that is, certain models or documents have been brought to a prescribed state. Milestones enable managers and developers to monitor the progress of the work as it passes these four key points. Over the four phases, five workflows that take place: requirements, analysis, design, implementation and test (see Figure 7). Work in all models continues over all phases, as indicated by the increased filling-in of the models (see Figure 8).

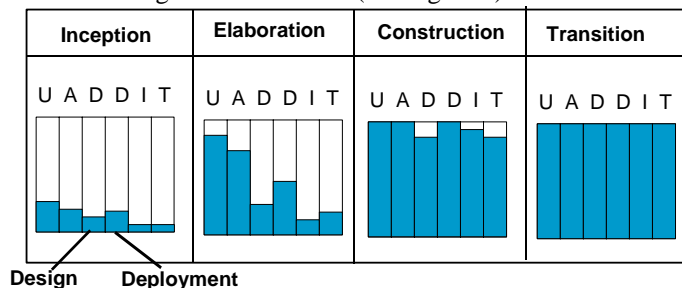


Figure 8 Evolution of models throughout phases

Figure 9 presents the evolution of models in terms of accomplished completeness over time. The values of completeness proposed are approximated from the curves

proposed in [24], and are given for illustrative purposes. These values may come from an experience database.

The model of deadline preservation (presented in the previous section) can easily be used for monitoring the evolution of each of the models throughout the four phases. The curves in Figure 9 represent the expected behaviour of the process. For each of the models, a monitoring for deadline preservation can be used in each of the phases.

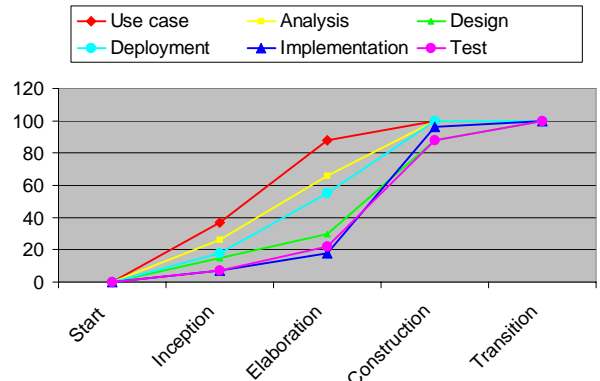


Figure 9 Evolution of models in terms of accomplished completeness

Let us consider the example of the inception phase, and the monitoring model will be used for measuring the confidence that the different product models will have achieved their expected completeness. The monitoring model is instantiated six times, for each one of the product models. The instantiated information *sourceAgent* is set for each one of them, indicating which are the agents that will provide this information (for instance for the designer for the design model). The *completenessName* is equally set for each one of the product models (for instance admitting that the completeness of the design model is named *designModelCompleteness*, in the instantiation for the design model monitoring, the *completenessName* will have the value *designModelCompleteness*).

At the end of the inception phase, the expected completeness for the different models are: use case around 37%, analysis around 26%, design around 15%, deployment around 18%, construction and transition around 7%. The inception phase is viewed as an activity (with a given allocated time), in which the completeness of the product models has to evolve. As the inception phase advances, reports on the completeness are made for the monitoring system, and the confidence in reaching the desired completeness by the end of the phase is computed for each one of the models. Thus, the manager can monitor the advancement of product models and see that they all develop as they should, i.e. as described by the evolution curves in a typical process. When the development of one of the models is not going as expected, the manager is warned by alerts and alarms. This gives the possibility to react if necessary before the phase ends.

V. RELATED WORK

The need for monitoring processes is by now required and the CMM with its forth and fifth maturity levels (as well as other evaluation and improvement methods like Bootstrap [21], Spice [17], Quality Improvement Paradigm [5]) is an indication of this state of facts.

Studying the systems proposed in the software process literature, the first observation with respect to the monitoring issue is that the scope and the features are very broad. All studied systems use certain means for data collection, but the analyses made on collected data as well as the scope vary from one system to another. The monitoring analysis employed range from providing no analysis at all, as in the Hakoniwa system [23], to the use of classification trees in Amadeus [24] and of statistics and probability in Balboa [11] as well as in a monitoring prototype experiment [6] at AT&T Bell Laboratories.

Some process support environments are instrumented for collecting data on the process without providing any methodology for analyzing the data. Typically the data collected represents the execution history of the process enactment. In such cases the interpretation of the history is not automated and relays considerably on persons that have a good knowledge on the process support environment used, persons which are able to select the significant data out of the entire collected data. In some process support environments, like Hakoniwa [23], there is a selection in the data collected, which is displayed giving for some aspects a global view of the process.

The object of measurement and analysis is in some systems the process while in others the products issued from the process. In this sense the Balboa system [11] as well as the monitoring prototype undertaken at AT&T Bell Laboratories [6] are process oriented, while the Amadeus [24] system has product oriented measurement and analysis.

Balboa [11] proposes analysis for process discovery and process validation. The process validation is defined as the detection and characterization of differences between a model of a process and the actual enactment of the process. The analysis is done once the execution is finished, thus no reconciliation of the ongoing process being possible.

In [6] it is described the prototyping of a monitoring experiment undertaken at AT&T Bell Laboratories. The prototype was meant to provide the instrumentation for a full monitoring experiment. Analysis was made to see how the performers spent their time in different identified states (working the process, documenting, waiting for a document, reworking, etc.).

The Amadeus system [24] offers measurement techniques and feedback in a process centered environment and automates the control and measure of evolving products. The process model can be instrumented for data collection. It provides a script language for analyzing the data collected.

All the studied systems make the assumption that precise and certain information is available. The prototyping experiment has some means of handling imprecision in the form of granularity, measurements are made on a daily basis, even though finer grain information would give a more accurate view of the process. This approach is meant to reduce experiment costs and to make it less intrusive. We agree that assuming the availability of precise information increases the costs of data collection, thus limiting the usage of the monitoring system that becomes more costly and intrusive.

None of the studied systems proposes a formal approach for monitoring. The systems are dedicated to a certain type of analyses.

The OMEGA environment provides a language for defining monitoring models as well as mechanisms for executing such models. It explicitly handles imprecise and uncertain information

with the use of fuzzy logic. The system handles precise data in the classic way, as classical sets theory is a sub-set of the fuzzy sets theory.

OMEGA's architecture allows its integration in a process support environment leading furthermore to a reduction of data collection costs, as most of such data can be automatically collected, without much interference in the subject process. The monitoring is open to evolution as new analysis techniques can be added.

VI. CONCLUSION

The OMEGA environment was developed in the framework of the ESPRIT IV LTR Project PIE (Process Instance Evolution), where we work with academic as well as industrial partners⁵. The model for deadline preservation presented is used in a scenario proposed by one of our industrial partners (Dassault Systems).

OMEGA provides a way of handling the deviations between process model and process performance using a fuzzy based monitoring and has the following characteristics:

- provides a language for defining monitoring models and mechanisms for executing such models;
- provides process measurement mechanisms;
- offers analysis techniques focused on the conformance of the process enactment to the process plan;
- handles imprecise and/or uncertain information;
- can be easily integrated in a process support environment.

REFERENCES

- [1] Alloui, S. Cimpan, F. Oquendo and H. Verjus (2000): Alliance: An Agent-Based CASE Environment for Enterprise Process Modelling, Enactment and Quantitative Control, *Enterprise Information Systems*, Joaquim Filipe (Ed.), Kluwer Academic Publishers, 2000.
- [2] Alloui, I., S. Cimpan, F. Oquendo and H. Verjus (1999): Software Agents for bringing cooperating software-intensive processes under quantitative control using fuzzy sets. In *Proceedings of Many Facets of Process Engineering MFPE'99, Tunisia, May 1999*.
- [3] Alloui, I., S. Cimpan, F. Oquendo and H. Verjus (1999b): Tuning a Fuzzy Control System for Software Intensive Processes via Simulations. In *Proceedings of the IASTED International Conference on Modeling and Simulation, Philadelphia PA, USA, May 1999*.
- [4] Alloui, I., S. Cimpan, F. Oquendo and H. Verjus (1999c): A Fuzzy Sets based Mechanism Allowing the Tuning of a Software Intensive Processes Control System via Multiple Simulations. In *Proceedings of the AMSE International Conference on Modelling and Simulation MS'99, Santiago de Compostela, May 1999*.
- [5] Basili V.R., G. Caldiera and H.D. Rombach (1994): The

⁵ The partners are Univ. Joseph Fourier, Univ. de Savoie, Dassault Systèmes, Xerox Research Center(France), The Univ. of Manchester, TeamWare (Great Britain), Politecnico di Milano (Italy), Univ. Dortmund (Germany).

- Experience Factory . *Encyclopedia of Software Engineering*, 1994, Wiley.
- [6] Bradac M., D.P. Perry and L.G. Votta (1994): Prototyping a Process Monitoring Experiment. In *IEEE Transactions on Software Engineering*, vol. 20, no. 10, October 1994.
- [7] Cîmpan S. (2000): OMEGA: un formalisme et un système pour le monitoring des processus dans le cadre des environnements de génie logiciel, *PhD Thesis*, 28 January 2000, University of Savoie at Annecy, France 2000.
- [8] Cîmpan, S. and F. Oquendo (2000): OMEGA: a language and system for on-line monitoring of software-intensive processes, ACM SIGSOFT Software Engineering Notes, July 2000.
- [9] Cîmpan, S. and F. Oquendo (1999): On the Application of Fuzzy Sets Theory on the Monitoring of Software-Intensive Processes. In *Proceedings of the Eight International Fuzzy Systems Association World Congress IFSA'99, Taipei, Taiwan, August 1999*.
- [10] Cîmpan S. and F. Oquendo (1998): Fuzzy indicators for monitoring software processes. In *Proceedings of the 6th European Workshop on Software Process Technology, EWSPT'98, London, September 1998*, Springer Verlag.
- [11] Cook, J. E. and A. L. Wolf (1994): Toward Metrics for Process Validation. In *Third International Conference on the Software Process, Reston, Virginia, USA, October 1994*.
- [12] Cugola G. (1998): Inconsistency and Deviations in Process Support Systems. *PhD thesis Politecnico di Milano, February 1998*.
- [13] Derniame J-C, A. B. Kaba and B. Warboys (1999): *Software Process: Principles, Methodology, and Technology 1999*, Springer Verlag, Lecture Notes in computer Science 1500.
- [14] Dowson M. and C. Fernström (1994): Towards requirements for enactment mechanism. In *Proceedings of the Third European Workshop on Software Process Technology, EWSPT'94, Villard-De-Lans, France, February 1994*, Springer Verlag, LNCS 772.
- [15] Dubois D. and H. Prade (1993): Possibilistic Logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 3, 1993* (D.M. Gabbay, Ed.), Oxford University Press.
- [16] Estublier J., N. Belkhatir, M. Nacer and W. Melo (1992): Process-Centred SEE and Adele. In *Proceedings of the 5th International Workshop on Computer-Aided Software Engineering, CASE'92, Montreal, Quebec 1992*, IEEE Computer Society Press.
- [17] European Software Institute *Spice Trials, Phase 1 Report, Technical Report ESI-1996-TR-002/01, January 1996*
- [18] Estublier J., S. Dami and M. Amiour (1998): APEL: a Graphical Yet Executable Formalism for process Modelling. In *ASE journal (Automated Software Engineering)*, vol. 5, Issue 1, 1998, Kluwer Academi Publishers.
- [19] Feiler P.H. and W.S. Humphrey (1993): Software Process Development and Enactment. In *Proceedings of the 2nd International Conference on Software Process 1993, pages 28-40, Berlin, 1993*, IEEE Computer Society Press.
- [20] Fenton N.E. (1994): Software Measurement : A Necessary Scientific Basis. In *IEEE Transactions on Software Engineering*, vol. 20, no. 3, pages 199-206, March 1994.
- [21] Haase V., R. Messnarz, G. Koch, H.J. Kugler and P. Decrinis (1994): Bootstrap : Fine-Tuning Process Assessment. In *IEEE Software*, pages 25-35, July 1994.
- [22] Humphrey W.S. (1988): Characterising the software process: A maturity framework. In *IEEE Software*, 5(2), pages 73-79, March 1988.
- [23] Iida H., K. Mimura, K. Inoue and K. Torii (1993): Hakoniwa: Monitor and Navigation System for Cooperative Development Based on Activity Sequence Model. In *Proceedings of 2nd International Conference on Software Process, Los Alamitos, California, 1993*, IEEE CS Press.
- [24] Jacobson I., G. Booch and J. Rumbaugh (1999): *The Unified Software Development Process, 1999*, Addison Wesley, ISBN 0-201-57169-2.
- [25] Selby R.W, A.A. Porter, D.C. Schmidt and J. Berney (1991): Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development. In *Proceedings 13th International Conference on Software Engineering. Los Alamitos, California. 1991*, IEEE CS Press.
- [26] Zadeh L.A. (1965): Fuzzy Sets. *Information and Control, Vol. 8, pages. 338-35, New York 1965*, Academic Press.