Abstract—Software product quality and project productivity require defining suitable software process models. The best process depends on the circumstances where it is applied. Typically, a process engineer tailors a specific process for each project or each project type from an organizational software process model. Frequently, tailoring is performed in an informal and reactive fashion, which is expensive, unrepeatable and error prone. Trying to deal with this challenge, we have built CASPER, a meta-process for defining adaptable software process models. This paper presents CASPER illustrating it using the ISPW-6 process. CASPER meta-process allows producing project specific processes in a planned way using four software process principles and a set of process practices that enable a feasible production strategy. According to its application to a canonical case, this paper concludes that CASPER enables a practical technique for tailoring a software process model.

Keywords—Software process lines; model-driven engineering

I. INTRODUCTION

Defining processes can improve the effectiveness of software development organizations [14]. Therefore, software process definitions are both useful for practitioners and reasonably economical to produce. Process engineering is the area involved with the practices for the definition, application, improvement and evolution of software processes [13]. Process engineering follows itself a process (a meta-process), a conceptual framework for expressing and composing software process models [21]. There is no standard software process suitable for all development situations since appropriateness depends on various organizational, project and product characteristics. So, each project requires a particular range of techniques and strategies, and selecting a set of practices and integrating them into a coherent process should also be aligned with the business context [12]. Due to both, high costs and time consumed, to define a specific process for each project is unfeasible. So, software processes need to be built for reuse and adaptation.

Families of software processes share common features as well as exhibit variability [30]. A Software Process Line (SPrL) is a special Software Product Line (SPL) in the software process engineering domain. A SPrL increases the reuse opportunities and it decreases the adaptation effort. Consequently, a process family is a promising approach for achieving a high level of reuse and tailoring [28][29][31]. In a SPrL, process domain engineering develops and maintains process assets promoting planned reuse instead of re-actively integrating unanticipated variability in the process model [3].

For the last six years we have worked in aiding small companies in Chile to define their software processes in an effort to improve national industry standard[1]. As part of this experience we have found some needs to specify context-suitable software process models. In a previous work, we have proposed a production strategy for tailoring a software process [16]. The strategy uses context information to apply rules that set the variation points to specific variants. However, the strategy by itself does not work if the process is not designed for adaptation and evolution. A general process model and a consistent delivered process model are achieved if an appropriate analysis of the commonalities and variabilities, and their relationships with the context are conducted. According to Ocampo et al. [23], a specific domain engineering meta-process should be conducted in order to arrange an adaptable software process model. Furthermore, to achieve a practical and automatic tailoring procedure, this meta-process must be applied in a disciplined, formal and scoped way, considering project contexts, process features, process architecture, and consistent production mechanisms.

This paper presents the entire SPrL meta-process: Context Adaptable Software Process EngineRing (CASPER) and its practices. Process domain engineering facilitates planned reuse anticipating variability in the process model [3]. The process adaptation effort is minimized in CASPER because it only needs to define a specific situation. The main problem for software engineers is to build software and not software processes, so adaptation should not be time consuming for software engineering stakeholders, and the software process models and software process tailoring rationale should be reused along different projects. We have been able to detail technical practices of CASPER based on our experience applying CASPER for defining a set of software process lines using parts of software processes in Chilean companies.

The paper is structured as follows. Section II presents the principles and the lifecycle of CASPER. A detailed explanation of the practices and their application in the ISPW-6 process is included in Sect. III. Related work is discussed in Sect. IV. Finally, some conclusions and further work are presented in Sect. V.

1ADAPTE (www.adapte.cl).
II. CONTEXT ADAPTABLE SOFTWARE PROCESS MODELS: CASPER

The planned software process tailoring approach of CASPER uses coherently three key approaches: software process lines, context and process modeling and MDE-based tailoring. CASPER has been built on a SPL process structure supported by a set of principles and practices.

A. CASPER Principles

The four essential principles of CASPER, derived from the state of the art, have led to consistently define the process and set of practices to achieve a useful approach:

Principle 1. Separation of Software Process Engineering and Software Engineering Domains: process modeling stakeholders and process acting stakeholders (project stakeholders) are usually not clearly distinguished [5]. Software process engineering stakeholders require different capabilities from those of software engineering stakeholders. In CASPER, the software process model definition task (including its adaptation mechanisms) is assigned to the process engineering group, and the process adaptation task (applying a tailoring mechanism) should be responsibility of the project management group. This does mean join work but responsibility separation, e.g., the process engineering group needs the software engineering group, to define the process, its variants and the application contexts.

Principle 2. Software Process Scoping: it is unclear which decision models can help determining which process can be applied to a specific project [5]. Particularly to SPRL, it is necessary to determine which processes and process elements should be part of the process line and which should not [5]. Scoping a CASPER SPRL will determine when the process will be used and which process elements (common and variable) will be required in each situation.

Principle 3. Software Process Models are Software Models too: similar to software models, process models can use an MDE approach for different goals: design, adaptation, simulation and enactment [1][13]. Using an MDE production strategy in CASPER, software process models and software process tailoring rationale can be reused in different projects to automatically produce specific process models.

Principle 4. Software Process Adaptation Complexity Hiding: process tailoring involves intensive knowledge and it is time consuming mainly when done manually [23][27]. A process modeling approach provides ways to cost-efficiently tailor a general process model into a project-specific process model [5]. In CASPER, the software process engineering group, according to the SPRL scope, defines an adaptable software process model, an adaptation context and a set of tailoring rules to simplify the process tailoring.

B. CASPER Subprocesses

The CASPER meta-process presented in Figure 1 embodies two main subprocesses:

Process domain engineering: is an iterative process focused on capturing software process domain knowledge and developing process model core assets (contexts, process features, process components, reference process models, tailoring rationale and concrete tailoring rules) enabling the implementation of each context-adapted process model.

Process application engineering: a context-adapted software process model is automatically produced according to the requirements of a specific project using a reference process model. The production strategy is an MDE implementation where the tailoring rationale is expressed as a set of transformation rules.

When CASPER is adopted, the organization should include two different work teams: one for domain software process engineering (Software Process Engineering Group - SPEG) and another for software process application engineering (Project Team - PT). The SPED is the same as in a Software Process Improvement (SPI) approach. The PT corresponds to people in charge of applying the software process to specific projects. The SPEG takes care of developing and evolving the process family. The PT produces and applies a family member according to the project-specific context.

C. CASPER Domain Engineering

Process domain engineering is an iterative process that captures the knowledge gained in the software process domain. As defined in Figure 2, it includes five activities: process context analysis, process feature analysis, scoping analysis, reference process model design, and production strategy implementation. These activities are performed by both, the SPEG and PT; however, the PT mainly participates as a source of requirements and experience. The process context analysis allows the understanding of process requirements. The process feature analysis identifies and defines process features that establish the general process elements and their classification as common, optional or alternative. Scoping validates the context characteristics, the process features and defines their relationships. The reference process
model design implements the process features in a reference process model using the variability and reuse mechanisms available in the process modeling language. The production strategy implementation defines and implements a set of decisions to adapt (tailoring rationale) the process model.

D. CASPER Application Engineering

The application engineering, depicted in Figure 3 defines a project specific context configuration model according to project information and executes the production strategy. If it is not possible to model the situation with the context model, the most approximated context configuration should be defined by the software engineering group. Then, the obtained model can be manually tailored by the SEPG. In this case, a scoping decision must be made by this group referring the SPlL (scope extension or no).

III. TECHNICAL PRACTICES IN CASPER

The main practices introduced by CASPER’s Domain Engineering are here explained and their associated work products are illustrated with the ISPW-6 process. This example consists of a core problem and several extensions. Solution to the core problem has been modeled as the common part of the process. The extensions provide rich variants used in CASPER to demonstrate its capabilities for defining contexts, process features and process models.

A. Software Process Context Analysis

The context of a project may vary according to different project variables such as: product size, project duration, product complexity, team size and application domain knowledge, among others. The context model is used in CASPER as one of the requirements artifacts in the process domain engineering. This fact is supported by the principle that the context should be determined before designing or comparing processes. First Context Elicitation identifies the information sources including past projects, process experts, previous process adaptations, identified kinds of projects, process templates, process configurations and tailoring guides. The elicitation could be conducted as a workshop, using brainstorming to identify context attributes, and then a review and prioritization could be performed, optionally a prototypical context model could be used. Other requirements engineering techniques could also be used and general sources of relevant context information can be found in [8][25]. In the ISPW-6 process, the elicitation was conducted interpreting its documentation. Some context attributes as enactable process, cooperation, level of the skills, resource constraints, tool availability and concurrence were identified using section 3, optional extentions. In practice not all combinations of context attribute values are relevant for adaptation. According to Tobias et al. [10], a process model must address only those combinations frequently occurring in practice. To extract these combinations, a survey or a workshop could be conducted. The result of this analysis is a group of context attribute values and specific combinations that are relevant. In ISPW-6 process and using an initial context model, a review process was conducted. The reviewer stated that, with respect to tool availability, there were no relevant aspects for tailoring the software process (it refers to an enacted process). The context model is formally specified as an instance of SPCM, as applied in [16]. The context model of the ISPW-6 process is depicted in Figure 4.
context includes three dimensions: project, process and team. Attributes Resources Availability Constraints, Enactable and Concurrence have possible values Yes and No. The possible values of the Skills context attribute are Homogeneous and Heterogeneous, and for the Cooperation context attribute the possible values are High and Low.

B. Software Process Features Modeling

Feature modeling was proposed as part of the Feature-Oriented Domain Analysis (FODA) method [17], and since then, it has been applied in a number of different domains [2]. Feature modeling is used to model the common and variable properties in product-lines. At early stages, feature modeling enables product-line scoping, i.e., deciding which features should be supported by a SPL and which should not. A feature diagram is a visual notation of a feature model, which is basically an alternative (OR, XOR) or inclusive (AND) tree of features where each sub-tree can be optional (represented by a white circle) or mandatory (represented by a black circle). Requires and excludes constraints between nodes can also be defined (represented by unidirectional or bidirectional arrows respectively). In CASPER software process features are considered as a special kind of software features. Examples of process features are process properties (life cycle type, maturity level), method elements (method fragments), process elements (process components, process fragments), process with method elements (chunks), method plug-in elements (reusable components, process patterns, processes and configurations), process packages, method packages and categories. Because a software process has multiple views, these perspectives such as role view, process view and work product view among others, could be defined in one or many process feature models. Elements in the same and different perspectives can include relevant constraints for a coherent process assembly, e.g. the requires constraint. In general a process feature could be implemented with many process pieces and a process piece could implement many features.

Figure 5 depicts the feature model where commonalities and variabilities of the process for ISPW-6 process example are specified. A single perspective (breakdown perspective) has been used, but different kinds of the elements are related following a top down decomposition from process components to method and managed content method elements. Schedule Revision is an optional process feature because it is necessary only for the initial schedule, as derived at planning; it would be modified to cope with actual results obtained from process enactment, i.e. the process model is not enactable. Also, the Resources Availability Analysis process feature is used only when these project constraints have been identified. Review Design process feature may be realized by two alternative non exclusive process features: a Design Inspection task and a Design Peer Review task. Furthermore Schedule and Assign Task process feature may be realized by two alternative exclusive process features: Basic Scheduling and Dynamic Scheduling tasks. The Dynamic Scheduling and Schedule Revision tasks must be present both or none. Each task feature includes the related method elements (roles and work products). The process features notation is much closer to the implementation language but implementation issues could be relaxed in the modeling.

C. Software Process Scoping

Product line scoping determines what is inside and what is outside of the SPL [11]. This identifies those context entities with which products deal (the context), and it also establishes the commonalities and sets limits on the variability
of the products. Getting the scope right is important: if the scope is too large, the core assets will have to accommodate so much variation that they will be too complex to be useful and cost-effective in any product. If the scope is too small, the SPL may not have enough opportunities to recover the investment in the core assets and having either a unique process or just a couple of processes would be a better strategy. And if the scope bounds the wrong products, the SPL will not satisfy the target market. A SPL may result useful when a reference process is involved but some manageable differences between projects are relevant for productivity and quality issues.

1) Process Model Scoping Definition: Armbrust et al. [3] define Software Process Scoping as the systematic characterization of products, projects, and processes and the selection of processes and process elements, so that product development and project execution are supported efficiently and process management effort is minimized. Software process optional and alternative features determine the possible process models. However it is important to determine for each process what the most suitable context is. A specific context configuration could determine a project specific process, but the relationship between processes and configuration contexts is a complex many to many relationship. Scoping in CASPER represents a preliminary way to define tailoring rules to obtain a suitable software process model. When the process line scope is defined, the relationship between context attributes and process features could be:

**One to one relationship:** is the simplest decision, a process feature variable depends on only one context attribute.

**One to many relationship:** a context attribute defines many process features. These context attributes are very relevant because congruence between the decisions of many related features is required, e.g., features related by the requires relationship. In this case many rules must be defined for the same context attribute.

**Many to one relationship:** this case is very complex. Some context attributes together define the process feature selection. This combination requires an order in a static decision rule and also a priority if required at tailoring time. It is also possible to define when an element must be selected or not by one specific combination of context attributes. A color code can be used, e.g., coloring green the cross between the involved context attribute values and the process feature if the process feature must be mandatorily selected for this attribute value, red if the context attribute value implies eliminating a process feature, and yellow when a combination of values of different context attributes defines to select or not a process feature.

**Many to many relationship:** this case is the most complex. The complexity of the previous case is increased by the complexity of the second case.

Figure 6 depicts a scope solution for the ISPW-6 process: rows are context attributes (and values) and columns are the variable features. In this case four one to one and a one to many relationships appear. For example the Cooperation attribute determines when the Design Review process feature is resolved. The case of Enactable attribute is one to many, because this attribute determines how the scheduling is implemented and if Schedule Revision is applied. A semaphore code has been used.

2) Scope Change: Similar to the SPL Framework of the SEI [11], the process scope definition is used during software process tailoring to know if the expected process model is a feasible member of the SPrL. Sometimes a process clearly falls within the defined scope and others it will be clearly out of scope. A more complex situation is when the process is located on the boundary. In that case, a scope analysis may help to determine if the organization requires this new process as part of the SPrL, and then the scope can be extended appropriately. In CASPER, the scope could be extended when adding a new context attribute or a new context attribute possible value. Also, when a new optional/alternative process feature is added or a new constraint on a context attribute-feature relationship is added, and also when a set of old context attributes, process features or a context-feature relationship is modified. For example, the team size starts to be relevant for selecting the Integration Testing process feature.

D. Software Process Modeling

According to Washizaki [31] a Process Line Architecture (PLA) is a process structure that reflects the commonality and variability in a collection of processes that make up a process line from the perspective of overall optimization. The overall optimization refers to preparing a PLA with general usefulness rather than defining separate but similar optimized processes. By deriving individual processes from the PLA, the fixed amount of additional effort required in the future can be reduced, and productivity can be improved. CASPER production strategy uses a PLA as an organizational
(or reference) process where commonality is represented by the core process (kernel process). Processes are modeled using SPEM [24]. Variability is represented by the variation points in the kernel process and process variants are available as plug-ins, process components, process patterns, processes, activities, method elements and managed content elements.

SPEM processes in CASPER follow the idea of a PLA with variation points that can be changed according to the particularities of a specific project. Process variants corresponding to optional or alternative process or method elements are applied to the variation points as shown in Figure 7 in a model transformation (generation). When a generative strategy is used, a set of rules relating variation points with project characteristics is required. These rules embody tailoring decisions that take place to different process granularity levels (both, optional and alternative).

Specific examples follow:
1) A complete process configuration (the template tailoring approach).
2) A complete process pattern could be selected to one discipline or an iteration.
3) A complete set of document templates according to what is rigorously required.
4) A complete method plug-in could be selected because specific technology methods or processes are required.
5) A complete method/process package or process component could be selected because a third party’s specific interface is required.
6) A complete phase could be selected according to a management criterion.
7) A medium-size activity could be selected according to a technical criterion.
8) A set of tasks, work products or roles could be selected according to a technical/management criterion.

To different granularity levels, variation point modeling depends on the feature models defined in the domain analysis, on the PLA and the constructs of the used software process metamodel. Thus, a process feature can be implemented by one element (e.g. a task definition), a set of elements (e.g. a set of Templates), a process pattern, or an activity, among others. The SPEM architecture provides a framework for structuring a software process model. The basic steps for structuring a process supporting variability are:

Step 1. Create a Method Library: A Method Library is the complete container of the organizational software process model and the adapted software process models.

Step 2. Identify common, optional and alternative method plug-ins: Create and reuse the respective method plug-ins. Relate the process features to the respective Method Elements. Define the variation points in the kernel process and their associations to variant method plug-ins.

Step 3. For each Method Plug-in create the respective process packages: Structure each process package according to the corresponding design decisions. Create and identify core and variant process patterns. Relate process features to the respective process elements (including process patterns).

For each identified alternative feature that is part of an Activity, a hierarchy of activity elements can be created using the variability types defined in SPEM. When an alternative feature has been identified in step 3 to a Method Element in Use, then the element with the variation point must be set to a default method Content Element. This link considers the variation point to be fixed at tailoring time. Similarly, the usedActivity link of an Activity to another Activity could be considered as an alternative (as in Figure 7) variation point (extended activity in SPEM). In this case this link must be set to a default Activity and fixed at tailoring time to the corresponding Activity identified as a variant.

A package oriented view of the ISPW-6 process model as an instance of SPEM is presented in Figure 8 where a method library, a method plugin and their elements (Packages, Patterns, Process Elements and Method Elements) have been organized. The tailored elements (white elements) are obtained from others by applying some tailoring strategy. The general ISPW-6 process has been modeled as a capability pattern, using tasks (simply called tasks) in use and organized to cover four main activities: Change Design, Change Testing, Schedule Planning and Monitor Progress. In Figure 9 the first three activities are detailed and connected whereas the last is detailed in Figure 10.
The Schedule and Assign Tasks task assigns resources to tasks and estimates the schedule required for those resources to carry out the tasks. This task has two exclusive alternatives: Basic Scheduling task and Dynamic Scheduling task. The Modify Design task modifies the design for the code unit affected by the requirements change. The Design Review task is the formal review of the modified design; it is conducted by a team including the design engineer who produced the design modifications. The Design Review task can be defined at tailoring time as Inspection Design task or Design Peer Review task. Inspection Design applies when there are senior and junior designers in the team and Design Peer Review when the staffs skills are all the same.

The Modify Code task involves the implementation of the design changes into source code, and its compilation into object code. The Modify Test Plans task refers to the modification of test plans and objectives to include testing of capabilities related to the requirements change prompting this software change; it is an optional task that entails the consideration of multiple changes: rather than considering only one change to the software, we assume that several changes can be developed independently and concurrently. The Modify Unit Test Package involves the modification of the actual unit test package for the affected code unit according to the modifications made to the test plans and objectives. When the full set of changes has passed their independent unit tests, they are integrated into a full system, and integration testing is conducted. Of course, integration testing may reveal problems in the units changed, or elsewhere in the system, and corresponding additional modifications could be needed to correct them.

Concurrently to the technical change process, the project manager performs the Monitor Progress activity for monitoring the change tasks. This activity includes the mandatory tasks Monitor Progress and Rescheduling Tasks, and the optional tasks Schedule Revision and Resources Availability Analysis. The Monitor Progress task involves monitoring progress and status of the work. This implies notifying the completion of each task as well as informal information. The Rescheduling Tasks task refers to deviations from the plan where rescheduling of tasks is required. The Schedule Revision task refers to the way the initial schedule derived at the beginning would be modified to cope with the results of process enactment. The Resources Availability Analysis task refers to the continuous analysis to cope with resource constraints; these might include unavailability or partial availability of personnel when a task is ready to be started.

E. MDE Production Strategy

Adaptation in CASPER is the process to resolve variation points to specific variants according to a context config-
The SPrL scope model is the main input to the transformation implementation because all process features have been related to context attributes (as preliminary tailoring rules). In CASPER, the project manager should only provide the characteristics of the project at hand during the application engineering, and a project-adapted process is rapidly and automatically generated.

In CASPER, tailoring decisions are encapsulated as auxiliary rules as depicted in Figure 11. Rules about tailoring the general process model are composed incrementally according to the scope model. So we can configure new process models through a generative strategy by recombining partial tailoring transformation rules and thus reusing the knowledge they embody. All and only the required roles, activities and work products will be present in the adapted process, and no extra work would be required. The adapted process is thus more efficient, and the tailoring process is more reliable as well, since it is computed automatically.

SPEM and SCPM metamodels have been defined as Ecore metamodels in EMF and the transformations have been implemented as ATL rules. Models were implemented as instances of defined metamodels and edited using Exeed (Extended EMF Editor), an extended version of the built-in EMF reflective editor that enables customization of labels and icons by adding annotations to Ecore metamodels. The CASPER tool implementation was developed in EMF 3.4 and the ATL plug-in 2.0. In CASPER, ATL transformations produce a delta of the input, the rules basically are copiers with small changes at the variation points, and helper functions are called to fix the variants according to a specific context configuration when a variation point is found. Variation points are located on copy rules whereas variants are recovered from helpers.

Each variation point identified and associated with a process feature requires one or more simple or complex rules. To make a decision, the scoping table aids to identify when a variant is selected. To set an optional variation point requires a rule returning a Boolean value, whereas to set an alternative variation point requires a rule returning a Kind value (according to the Kind defined in the selection). For instance, an alternative variation point of kind Task requires a rule returning a Task value. In the case of the ISPW-6 process, the alternativeToSchedule rule for the Schedule and Assign Task task in use has been implemented using an auxiliary rule (helper ATL) called from the matched rule (Task Use). This rule is shown in Figure 11. Provided that we use SPEM variability mechanisms, a process element (e.g. Schedule and Assign Task) could be linked to several variants of method elements (e.g. Basic Scheduling) according to the Enactable Context Attribute Value. Therefore, we define an AlternativeRule as a rule that returns the selected method element according to the helper rule logic. The AlternativeRule chooses the most suitable TaskDefinition variant according to the Context Attribute Value in the context. If there were more variability points, a combination of rules would be applied. Additionally, if priorities to make trade-offs were required, the respective helpers would be used. Executing the rules to a context project including an enactable process, a homogeneous and cooperative teamwork and no concurrent changes and with resource availability constraints, the tailored process looks as shown in in Figures 12 and 13.

F. Discussion

CASPER meta-process shows a logical path to build generic but adaptable software processes. Tasks follow a general approach of software product lines whereas artifacts follow a specific model-based implementation. The solution is simple and yet practical and appropriate for the software industry. The main difficulty of the approach is the required formalization because the process model goes beyond traditional process elements: a CASPER SPrL also includes context models, a well defined scope, and formalized tailoring rules.

Nevertheless, in practice a few context attribute values and variants are enough to produce a limited but sufficient number of software processes, that formally defined enable to achieve a systematic adaptation. The tasks of CASPER

---

3CASPER tool website https://sites.google.com/site/softwaremetaprocess/  
4EMF website http://download.eclipse.org/tools/emf  
5ATL website http://www.eclipse.org/downloads/
A set of processes can be defined and associated with a set of context configurations. A specific experience is presented in Bohem’s Incremental Commitment Life Cycle Process [9] where a set of decision points based on risks are defined. For this model a set of patterns for Rapid-Fielding Projects has been developed by Koolmanojwong et al. [19]. A pattern is selected according to the situation. As each organization is unique, the process model is also unique, so a more general approach than this one is required, but a process or pattern selection is less flexible; process models are limited and complex to maintain. So, when the features tree grows it is critical to produce each process model. In contrast, partial decisions about each feature according to specific situations facilitates producing only one and the most suitable process to this configuration context.

Lee et al. [20] propose a method for scoping SPLs in which use contexts are related to product features similar to CASPER. However, CASPER is simpler because the process features are directly related with the context information using a mapping table or other mechanisms as model mapping or abstract rules. This direct relationship enables to directly define the tailoring rules, and so, an automatic strategy to produce software process models. CASPER could be supplemented with PuLSE-Eco, a practical method used for determining the scope of a SPL [6]. In CASPER tailoring knowledge is reused too and it is increased as the SPrL evolves instead of reusing only software process features as the different approaches identified by [15].

V. CONCLUSIONS AND FURTHER WORK

In this paper we have presented a set of practices defined as part of CASPER, a meta-process for building SPrL. These practices include context modeling, process feature modeling, scope determination, process modeling and production strategy implementation. Each practice has been exemplified using ISPW-6 process. The production strategy is the core of CASPER, however this strategy is insufficient if the SPL does not include suitable structures to reuse process fragments in a consistent way with the whole process. Context modeling enables the production strategy because it requires contextual information to make tailoring decisions. Process feature modeling enables to determine where variation points must be included and what variabilities are required. Scoping analysis allows early discovering the relationships between context and process features, and so determines in an abstract way what the process family is. Process modeling produces a generic software process, including its variants, to generate a set of consistent software processes. As part of our ongoing work, we are integrating all practices following the MDE approach: the scoping model and the general process model will be used to automatically generate the tailoring rules. In fact, the scoping model counts on the abstract information to define the rules for any implementation language. If the rules

IV. RELATED WORK

According to Aharoni et al. [11] a specific context can be defined as a vector of characteristics that relate to the organization, the project, the developing team, the customer, etc. However, defining the context as a formal model enables us to automatically tailor the organizational process according to it. The SPCM (Software Process Context Metamodel) has been previously defined by Hurtado et al. [16] in order to express both context models at the organizational level and the context configuration models at project specific level. However, this work does not define a specific practice to elicit and analyze a context model.

Figure 13. Monitor Progress Tailored to a Specific Context

...
are generated, defining the production strategy will become simpler than the technique presented in this work.

Acknowledgements
This work has been partly funded by project Fondef D09I1171 of Conicyt, Chile.

References