An Execution Platform for Extensible Runtime Models

Mario Sánchez*[1][2], Iván Barrero[1], Jorge Villalobos[1], and Dirk Deridder[2]

[1] Software Construction Group, Universidad de los Andes, Bogotá, Colombia
{mar-san1,iv-barre,jvillalo}@uniandes.edu.co,
[2] System and Software Engineering Lab, Vrije Universiteit Brussels, Belgium
Dirk.Deridder@vub.ac.be

Abstract. In model-driven development, high level models are produced in the analysis or design phases, and then they are transformed and refined up to the implementation phase. The output of the last step usually includes executable code because it needs to introduce the details that are required for execution. However, some explicit structural information is lost or scattered, making it difficult to use information from the high level models to control and monitor the execution of the systems.

In this paper we propose the usage of a platform based on extensible, executable models, which alleviates the loss of information. When these models are used, the similarity between the structure of high level models and of the elements in runtime eases the construction and usage of the system. Moreover, it becomes possible to build reusable monitoring and control tools that are only dependent on the platform, and not on the specific applications. Our proposal is shown in the specific context of workflow-based applications, where monitoring and control is critical.

1 Introduction

In many applications, runtime information is necessary for a variety of reasons. For instance, it might be because it is necessary to take swift corrective actions during execution, or because historical data is required to check and improve performance. Runtime information might be difficult to manage and interpret, and thus it is desirable to have powerful tools to handle it. Furthermore, since such tools are difficult to build and maintain, then it is desirable to have reusable tools that can serve to monitor and control different applications.

Additionally, these tools should also be capable of managing high-level concepts. Modern systems allow business experts to have more direct control over the applications, instead of relying on technology experts as in the past. Because of this, it is expected for those applications to offer information in terms of high level business concepts. Similarly, the control interfaces should offer high level

* Supported by the VLIR funded CARAMELOS project http://ssel.vub.ac.be/caramelos/ and by Colciencias

operations instead of only low-level operations that require technical knowledge about implementation details.

One possible alternative to manage these requirements, is to build model-based tools to do runtime monitoring and control of applications. Given the flexibility offered by unsing models, such tools would be reusable and have the ability to manage high level concepts and information. However, the feasibility of building such tools depends on features required in the monitored applications. As we will see in this paper, these features are not always available.

Many platforms used to execute applications today have limitations. In the first place, the interfaces offered to gather runtime information or to control the execution of the applications are nor standardized; they do not even have commonalities among several applications. Thus, it is difficult to have reusable monitoring tools that manage information beyond the virtual machine or the operating system level. Furthermore, the interfaces are limited in the quality of information offered, which can be insufficient to adequately control the execution.

Another problem is that runtime information might be difficult to interpret in terms of business concepts. This happens because the mapping from implementation elements to high level concepts can be difficult to establish. As shown in figure 1, the analysis and design artifacts, which contain all the relevant business information, are somehow used to produce implementation artifacts. Depending on the strategy used, this step can produce different kinds of traceability information that can be later used to reconstruct the transformations. This figure also shows that monitoring and control tools are between the business and the implementation level. The functionalities of these tools depend on low level implementation data, events, notifications, actions, etc. However, the users that use the tools expect to see all those low level elements in terms of the corresponding business concepts. How to make this translation is thus the problem that has to be solved.

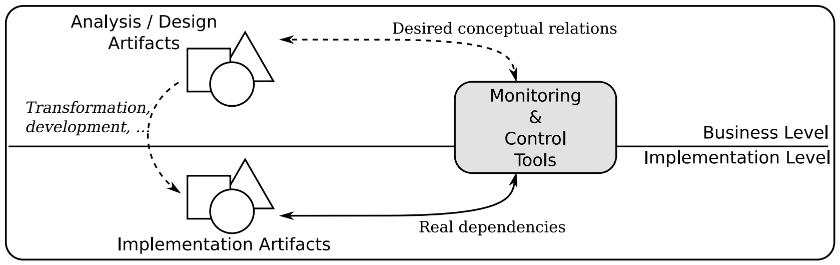

Analysis / Design Artifacts
Desired conceptual relations
Transformation, development, ...
Monitoring & Control Tools
Business Level
Implementation Level
Real dependencies
Implementation Artifacts

Fig. 1. Dependencies of monitoring tools.

One example of the previous ideas could be an application where *employees* have to perform a certain number of *activities*, and someone wants to monitor the percentage of activities finished by each employee in a given period. In this case, employee and activity are very precise business concepts, but they can be scattered in the implementation. For instance, to retrieve the required performance information it might be necessary to get low level information about sessions, services invoked, database accesses and other. One alternative to preserve this information could be to keep traceability information. However, there are other kind of problems associated to the complexity of keeping track, managing and interpreting this information.

The proposal presented in this paper targets the creation of platforms that support the execution of a wide range of applications and offers the necessary features to develop reusable monitoring and control tools with the requirements discussed previously. In the first place, this proposal advocates for the development of model-based applications. In second place, it proposes a reusable platform for model execution, where runtime information is easily obtainable. Furthermore, since the elements in execution have a structure that is very close to the structure of elements in design, then it is easy to establish a mapping between implementation and business elements.

This paper focuses on presenting the Cumbia platform. This platform is based on extensible, executable models and it offers several advantages to runtime monitoring and control of the applications that are executed on it.

Nowadays, workflows and wokflow-based applications are very important and new ones are built permanently. Furthermore, in this context monitoring and controlling are critical. In this paper, the Cumbia platform is illustrated in the workflow context, but its advantages are also explained for more general contexts.

The structure of the paper is as follows. Section 2 presents the problem of monitoring and controlling runtime systems and, in particular, workflows. Afterwards, section 3 presents the details of our proposal, and section 4 shows how it eases the runtime monitoring, management and adaptation of model-based systems. Finally, some related works and the conclusions are presented.

2 Runtime monitoring and workflows

As we discussed previously, in many applications it is necessary to have access to runtime information that can be used, for example, to support runtime decision-making. Additionally, to have useful and powerful monitoring and control tools it is necessary to have the means to raise the level of abstraction of the available runtime information. This raise allows users of the tools to make analysis and decisions from the business perspective instead than from the implementation point of view. For instance, a non technical user might prefer to know that the level of service offered by a partner application has dropped below the limit specified in a contract, instead of having to understand reports about timeouts or communication failures.

Part of the complexity associated to raising the level of abstraction is to be found in the implementation of the applications. In many cases, applications have structures and architectures that are very different from the structure of the problems that are supposed to be modeled and solved. Thus, it is difficult to reconcile low level runtime information with high level concepts, as required by monitoring and control tools. This problem, as well as the limitations in the interfaces to gather low-level information, becomes even more critical when monitoring requirements appear late in the life-cycle of the applications.

Based on the promises made by approaches like MDA [1] or xUML [2], model-based applications should face less problems to accommodate monitoring and control tools. However, in these approaches not all the discussed problems are solved because the last transformation or compilation step has as output executable code, which consistently scatters or loses some information that was originally available on the models. One approach that can be applied to overcome this problem inolves the usage of transformation models and traceabiliy information [3]. Although this might solve the inmediate problem of the lack of information, it creates other problems related to the interpretation of the information.

In the context of workflows and workflow-based applications these problems are also present. In the first place, in this context models are widely used by business experts, which use domain-specific languages to describe business processes taking into account the so-called business rules. Afterwards, these processes are deployed into workflow engines to be executed. At runtime, process designers expect to see the same concepts that they used in the definition. In order to make decisions, they have rules and policies that are based on that kind of information.

Normally, the execution of the processes is monitored and controlled either with low-level applications, such as engines' consoles, or with tools such as BAMs (Business Activity Monitoring). In general, BAMs are rather flexible and configurable, and they have as main goal raising the level of abstraction of execution information in order to make real time measurements of certain Key Performance Indicators (KPI) described by domain experts. However, BAMs are also tightly coupled to workflow engines' implementation, and to the workflow definition languages. As a result, the same BAM cannot be used with different engines. Moreover, even small changes to an engines' implementation, or to a language, can render unusable an entire BAM. This is something critical in this context because business rules and processes tend to evolve rapidly and the tools have to evolve along.

Finally, an increasingly important requirement in workflow applications is the ability to handle dynamic adaptation of the processes. This means that depending on internal or external events, the structure of processes might change at runtime, either in an autonomic fashion or following instructions specified by business experts. These changes have an impact on monitoring because the tools have to adapt and reflect the modifications. Furthermore, the tools used to describe the modifications should be integrated with the tools to do the monitoring and they should share the same high level language and concepts. It is expected that whoever directs the dynamic adaptation uses runtime information to make the necessary decisions.

3 Extensible executable models

The proposal presented in this paper is part of the Cumbia project of the Software Construction group of the Universidad de los Andes. In this project, we have developed the Cumbia platform for extensible, executable models. Originally, we developed this platform with the main goal of supporting extensible workflow-based applications, which might include complex monitoring requirements. Nevertheless, the platform is sufficiently generic and offers benefits that can be valuable in other contexts as well.

From the point of view of monitoring and control, a very important feature of the platform is the usage of executable models, which keep during execution all the structural information of the models. Since there is no loss of information, it is easier to rise the level of abstraction; in this case, creating the mapping between implementation and design elements is trivial. Another advantage of the platform is that it offers runtime information about every object in the model, that can be easily consulted from external applications. Moreover, the platform offers interfaces that can be used to easily integrate other applications, such as monitoring tools.

In order to support the execution of models, the platform manages the corresponding metamodels. The platform is very flexible in the support for metamodels, and also in the support for changes to the metamodels: they can grow and evolve without any significant impact to the platform. In section 3.1 it will be shown that the only requirement for metamodels is that they should be defined in terms of *open objects*, which are the special kind of executable elements that we defined for our platform. In this section, we will first briefly present XPM (eXtensible Process Metamodel), which is a simple metamodel for the definition of workflow processes that can be used in our platform. Afterwards, the main details about open objects will be discussed with the goal of explaining how monitoring and control are supported. More details about open objects can be found in [4].

3.1 XPM and Open Objects

The purpose of this section is to present one sample metamodel we have developed for the platform; we do not pretend to argue here about the completeness of the metamodel or its suitability to represent workflow processes. In order to present XPM we will use the sample process shown in figure 2, which was taken from the context of workflows for financial services. It defines a sequence of steps to study and approve a credit request. This process is initiated when a customer applies for a credit. Then, it requires an in-depth study of the submitted request and of the financial history of the customer. Finally, someone has to approve or reject the request based on the results generated by both studies.

This particular process shows most elements of the XPM metamodel. The process is composed by four *activities* that are connected through *ports* and *dataflows*. Each activity has a distinct *workspace* and each workspace executes

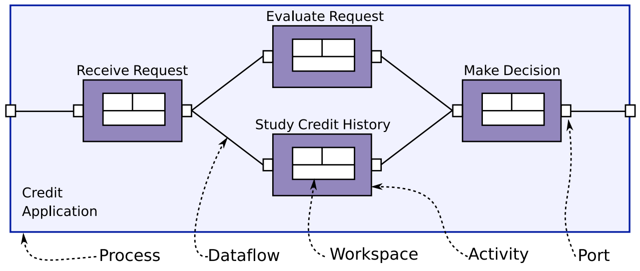


Fig. 2. A sample XPM process

a specific atomic task; activities serve to enclose them and handle all the synchronization and data management issues.

Every metamodel in Cumbia is based on something that we have called open objects, and thus they are the base for our platform. Every metamodel that is to be executed in our platform, has to be defined using open objects as its building block. As it will be shown, this means that every element in the metamodel has to be defined as a specialized open object. The workflow engine that we use to execute XPM processes was built using this approach and it has the open objects platform at its base.

The fundamental characteristic of an open object is that it is formed by an *entity*, a *state machine* associated to the entity, and a set of *actions*. An entity is just a traditional object with attributes and methods. It provides an attribute-based state to the open object and in its methods part of its behavior can be implemented. The state machine materializes an abstraction of the life-cycle of the entity, allowing other elements to know its state and react to its changes. Finally, actions are pieces of behavior that are associated to transitions of the state machine. When a transition is taken, its actions are executed in a synchronized way.

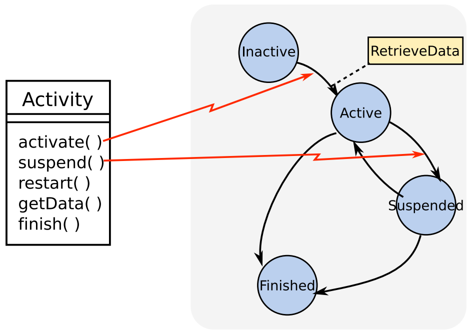


Fig. 3. The specialized open object that represents XPM activities.

The interaction between open objects is based on two mechanisms: event passing and method calling. In the specification of a state machine, each transition has an associated source event: when that event is received by the open object, that particular transition must be taken. This mechanism not only serves to synchronize open objects, but also serves to keep the state machine consistent with the internal-state of the entity: each time the latter is modified, it generates an event that changes the state in the automaton. Finally, other events are generated when transitions are taken and it is thus possible to synchronize open objects using state changes. The other interaction mechanism, method calling, is synchronous and is used to invoke the methods offered by the entities of open objects. These entities can receive method calls from two sources: there can be calls coming from external sources, such as user interaction or other related applications; additionally, the actions associated to transitions usually invoke methods of other entities, and thus they play a central role in the coordination of the entire model.

Figure 3 shows an open object that has been specialized to behave as an XPM activity. It has four states, and the state machine changes of state because of events generated by the entity or by other open objects. For instance it goes from the state `Inactive` to the state `Active` whenever the method `activate( )` is called, which in turn generates the event that moves the state machine. In addition, when this transition is taken, the action called `RetrieveData` is executed.

Several features of the platform and of the open objects facilitate the interaction with monitoring and control tools. In the first place, the platform is metamodel-based and changes to the metamodel can be seamlessly supported. This makes the platform particularly suitable to handle applications used in rapidly evolving contexts. In addition, open objects expose their state through the state machines, and this is an advantage for monitoring because the amount of available information. Furthermore, the interfaces provided by the platform offer powerful ways of interaction from external applications: it is possible to capture events to receive notifications, and it is also possible to invoke methods of the entities to control them (see figure 4). Another big advantage is that the mechanism of actions offers something similar to the interception meta-space described for the Lancaster Open ORB project [5, 6]: since actions can be installed and removed at runtime, it is possible to introduce extra-behavior between the processing of interactions. This additional behavior can be used to add further interactions with the monitoring applications. Finally, another big advantage offered by the platform is the inspection interface that allows to navigate the complete structure of the models, from the root level element (a process in XPM) to the last action or event.

Besides building the XPM engine, we have developed other metamodels and their corresponding "engines". For instance, we have built the metamodels and the engines for BPMN [7] and BPEL [8]. In order to use the platform for them, the first step was to design the metamodels and implement the required open object specializations. Since the platform itself is responsible for managing the instantiation and execution of the models and their open objects, then some

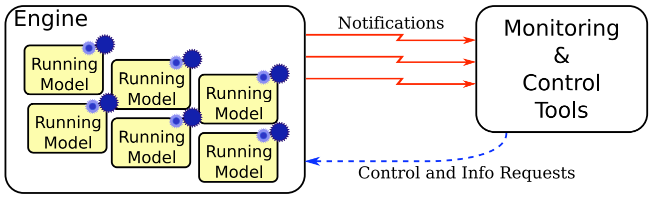


Fig. 4. Monitoring runtime models.

other specific services had to be defined for the engines in an ad-hoc way. For instance, in the case of the BPEL engine, the web-services based interface had to be specially developed.

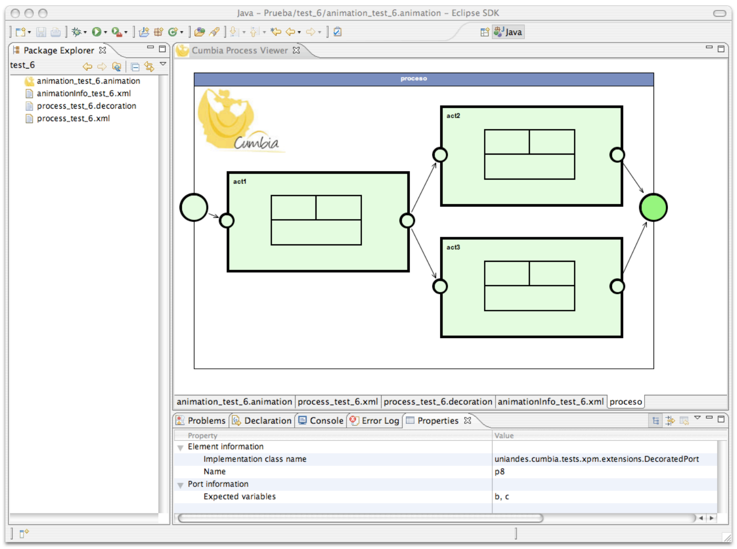
4 Monitoring and control

By taking advantage of the features of the platform described, it is possible to build very powerful and reusable monitoring applications. The combination of using explicit metamodels and the existence of a single executable platform leads to monitoring tools that are highly configurable. Since these applications are only dependent on the platform, they can be easily adapted for the usage with systems based on other metamodels that can be executed in the platform.

When using the platform, monitoring tools can manage and offer four kinds of information about the running applications. The first kind represents the structure of the models, which normally is fairly static. The second kind is information about the current state of the elements in the models. The third kind is historical information about the state of the elements in the model, that is the trace or the history of the execution. In these three cases, we are dealing with elements available in the model, and accordingly to what was said previously, this means that we are dealing with high level concepts that are put into execution. Similarly, although all the notifications received from the platform are low level, they could be transformed and interpreted as high level notifications. For instance, a notification about a transition taken in the state machine of an activity, might be transformed into a high level notification about the completion of the activity.

The fourth and last kind of information that can be monitored is composed by derived information, which is not directly part of the model or its execution, but can be calculated. This derived information has to be defined for each context, including the rules necessary to calculate it using the information provided by the platform. To define, manage and analyze this kind of information, it is useful to have model-based monitoring tools, where the definition of the relevant information can be easily made. In the context of workflows, a possible example of derived information would be the average time required to execute the activities of a process. Another example, would be the name of the employee that performed more activities in a given month.

We have developed some examples of applications that monitor the execution of applications based in open objects by managing the first three kinds of information described. The most basic of those applications is an open objects viewer. For a given open object, this application allows the observation of the structure of the state machine and shows its state changes.



Property	Value
Element information	
Implementation class name	uniandes.cumbia.tests.xpm.extensions.DecoratedPort
Name	p8
Port information	
Expected variables	b, c

Fig. 5. Screenshot of the XPM Viewer.

Another application that we have developed is a viewer for XPM processes (see figure 5). This application not only shows the structure of the processes that are running inside the XPM engine, but also animates the elements shown by changing the color of the activities that are executed. This application has other two interesting characteristics. The first one is the possibility of using a domain specific language that describes what the viewer has to do when it receives notifications about state changes in XPM elements. Although the language itself is not very powerful (it only allows some basic stuff like changing the color figures based on types and state changes), it gives some flexibility to the viewer and turns it into an example of a simple configurable monitoring tool for open objects.

The second characteristic is that this application was developed as decoupled from the XPM engine as possible. As a result, the XPM engine ignores totally the existence of the viewer, while the viewer only has dependencies towards the open objects platform and the XPM metamodel (not the XPM engine).

The most complex monitoring application that we have built for the platform is called the “Cumbia Test Framework” (CTF). Although users do not interact with the tool, this application observes the execution of the models and interacts with them following the instructions in a script. Afterwards, the CTF observes the execution, receives notifications and analyzes the traces to validate assertions. Moreover, the control statements in the script are described using a high level language. We have used the CTF to test the implementation of several metamodels, and in each case, the required specializations to the CTF have been minimal.

5 Conclusions

In this paper we have discussed about the importance of runtime monitoring and control and we have identified some useful requirements for monitoring applications. Among these requirements, the one that creates most of the implementation problems, is the need of giving high level information to the users of the tools, instead of providing implementation level information. Other problems that we explored were the limitations on the quantity and quality of the available runtime information, and the limited possibility of reuse for monitoring tools.

The proposal that we presented in this paper has two parts. First, it advocates for the use of model-based development techniques. Then, it proposes the usage of a platform based on extensible, executable models. This proposal has several advantages that, in the paper, were illustrated in the context of workflows. Among these advantages there is the usage of explicit metamodels, and the ease of integration with other applications besides control and monitoring tools.

The various benefits offered by the proposed platform are useful for the construction of monitoring and control tools. In particular, it makes possible the development of new, reusable monitoring tools that can be applied to several contexts which are to be executed on the platform.

References

1. Joaquin Miller and Jishnu Mukerji. MDA Guide. Object Management Group, Inc., Version 1.0.1., June 2003.
2. Mellor, S., Balcer, M.: Executable UML A Foundation For Model-Driven Architecture. Addison-Wesley, Indianapolis (2002).
3. Jouault, F.: Loosely Coupled Traceability for ATL In: Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability, Nuremberg, Germany (2005).
4. Sánchez, M., Villalobos, J.: A flexible architecture to build workflows using aspect-oriented concepts. In: Workshop Aspect Oriented Modeling (Twelfth Edition), Belgium (2008)
5. Blair, G. S., Costa, F. M., Saikosky, K., and Clarke, N. P. H. D. M. (2001). The design and implementation of Open ORB version 2. IEEE Distributed Systems Online Journal, 2(6).
6. Costa, F. M., Provensi, L. L. and Vaz, F. F.: Towards a More Effective Coupling of Reflection and Runtime Metamodels for Middleware. In: Proceedings of Models@Run.time 2006, Genova, Italy (2006).
7. BPMN Information, http://www.bpmn.org/
8. BPEL Specification, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf