Visualizing and Assessing a Compositional Approach of Business Process Design

Sebastien Mosser[1], Alexandre Bergel[2], Mireille Blay–Fornarino[1]

[1] University of Nice Sophia – Antipolis
CNRS, I3S Laboratory, MODALIS team
Sophia Antipolis, France
{mosser,blay}@polytech.unice.fr
[2]Department of Computer Science (DCC)
University of Chile, Santiago, Chile
www.bergel.eu

Abstract. In the context of Services Oriented Architecture (SOA), complex systems are realized through the design of *business–driven processes*. Since the design of a complete process can be very complex, composition tools such as *aspects* and *features* propose to define large systems by composing smaller artifacts (more easy to understand) into a complex one. But these techniques shift the system complexity into the definition of composition directives able to build it. At composition time, process designers needs to be supported to understand and assess their designed systems. We propose in this article a set of visualizations to represents *composition* and then identify patterns and categorization. We use the ADORE framework as underlying composition platform. We validate this work by presenting in this article instance of these visualizations obtained from a Car Crash Crisis Management system (CCCMS, a comparison referential for Aspect Oriented Modeling techniques). We use these visualization to assess the CCCMS realization.

Note for the proceeding reader: this paper makes use of colors. Although not mandatory for its understanding, an online (colored) version of this paper will ease the reading.

1 Introduction

An application that follows the Service Oriented Architecture paradigm (SOA, [1]) is an assembly of services that realizes business processes. Business processes are defined by business specialists and typically involve many services that are composed in a variety of ways. Furthermore, the need to extend a SOA application with new business features (to follow market trends) arises often in practice. In the technological context of Web Services, business processes can be implemented as *orchestrations of services* [2]. Existing tools and formalisms related to business processes (*e.g.* BPMN notation [3], BPEL industrial language [4]) are essentially technologically-driven. They use a *design–in–the–large* approach and

do not intrinsically provide language constructions and frameworks to support the introduction of new features into existing processes.

New paradigms such as aspects [5] and features [6] model application in terms of composing smaller units. Assuming that a complex system is difficult to understand by humans, they propose to reduce the complexity by defining several smaller artifacts instead of a single and large one. They identify and encapsulate parts of models that are relevant to a particular concern. A same feature may be shared and integrated into several processes simultaneously. These artifacts are then composed to produce the expected system. These approaches help taming the complexity of business processes design. As a consequence, the intrinsic complexity of the system is shifted into the composition directives used to build it. When a system involves many processes, it is necessary to have a holistic point of view on features, compositions and business processes to grasp it. In this paper we examine visualization method to tackle complexity of compositions at the application level.

The paper makes the following contributions and innovations:

– a number of visualizations dedicated to support designers when they define business processes using a compositional approach.
– benefits and design weakness are revealed using a number of patterns to assess compositions quality.
– scalability of the approach is sketched by using a very large case study as running example.

We motivate this contribution by presenting in section 2 our running example. Visualizations used to represent and assess compositions are then described in Section 3. Section 4 briefly presents implementation details. We propose a discussion on the approach benefits (associated to interesting perspectives) in Section 5. Finally, Section 6 describes an overview of related work, and Section 7 concludes this paper.

2 Running Example: Realizing a *CCCMS* using ADORE

This section presents the running example used to validate the visualization approach defended in this paper. It also presents the composition framework we used to realize this example, and highlights our identified needs of visualization.

2.1 Case Study: A Car Crash Crisis Management System

In Kienzle *et al.* [7], authors propose a common case study (a Crisis Management System, CMS) to compare existing *Aspect Oriented Modeling* approaches between each other. We consider this case study as a reference, and use it as a running example to illustrate the problematic tackled in this paper and the contribution we made. According to the definition given by this case study, a CMS is *"a system that facilitates coordination of activities and information flow between all stakeholders and parties that need to work together to handle a crisis"*.

Many types of crisis can be handled by such systems, including terrorist attacks, epidemics, accidents. To illustrate the case study, they provide an instance of a CMS in the context of car accidents. They define this system as the following:

> "*The Car Crash CMS (* CCCMS*) includes all the functionalities of general crisis management systems, and some additional features specific to car crashes such as facilitating the rescuing of victims at the crisis scene and the use of tow trucks to remove damaged vehicles.*"

The requirement document defines ten use cases, described using textual scenario. Each scenario defines first a *main success scenario* which represents the normal flow of actions to handle a crisis (*e.g.*, retrieve witness identity, contact firemen located near to the crash location). Then, a set of *extensions* are described to bypass the normal flow when specific actions occurs (*e.g.*, witness provides fake identification, firemen are not available for a quick intervention).

2.2 Composition framework: ADORE

The ADORE framework defines a compositional approach to support complex business processes modeling, using the orchestration of services paradigm. Models describing business–driven processes (abbreviated as *orchestrations*, defined as a set of partially ordered activities) are composed with process fragments (defined using the same formalism) to produce a larger process. *Fragments* realize models of small behavior and describe different aspects of a complex business process. ADORE thus allows a business expert to model these concerns separately and then compose them.

We only provide in this section an informal description of the techniques used in ADORE to support the composition. Using ADORE, designers can define *composition units* (abbreviated as *composition*) to describe the way fragments should be composed with orchestrations. The merge algorithm used to support the composition mechanism [8] computes the set of actions to be performed on the orchestration to automatically produce the composed process. Interaction detection mechanism helps designers to build reliable processes as output of the composition framework. Implementation details, environment screenshots and video demonstrations are available on the project web site[1].

We proposed in our previous work [9] a realization of the CCCMS system using ADORE. We realized all the use cases main scenarios as orchestration of services, and extensions as fragments to be integrated into these orchestrations. The complete set of designed models (12 orchestrations & 24 fragments, representing 196 activities scheduled by 224 relations in terms of implementation) is available on the CCCMS realization web page[2]

[1] http://www.adore-design.org
[2] http://www.adore-design.org/doku/examples/cccms/

2.3 Need for Visualization Techniques

When developing a large system, designers handle a large set of initial orchestrations, and a large set of fragments to apply in these orchestrations. Adore tackles the complexity of integrating fragments into orchestrations, at the activity level. As a consequence, designers can extract from the tool very detailed and fine–grained information about the effect of their fragment application. It results into an obscrure set of composition details, as shown in Fig. 1. This figure represents[3] one of the ten use–case driven composition extracted from the Cccms requirement document. It represents 7 fragments used in the context of this composition, and the different unification computed by Adore in order to perform the integration of these fragments into an orchestration.

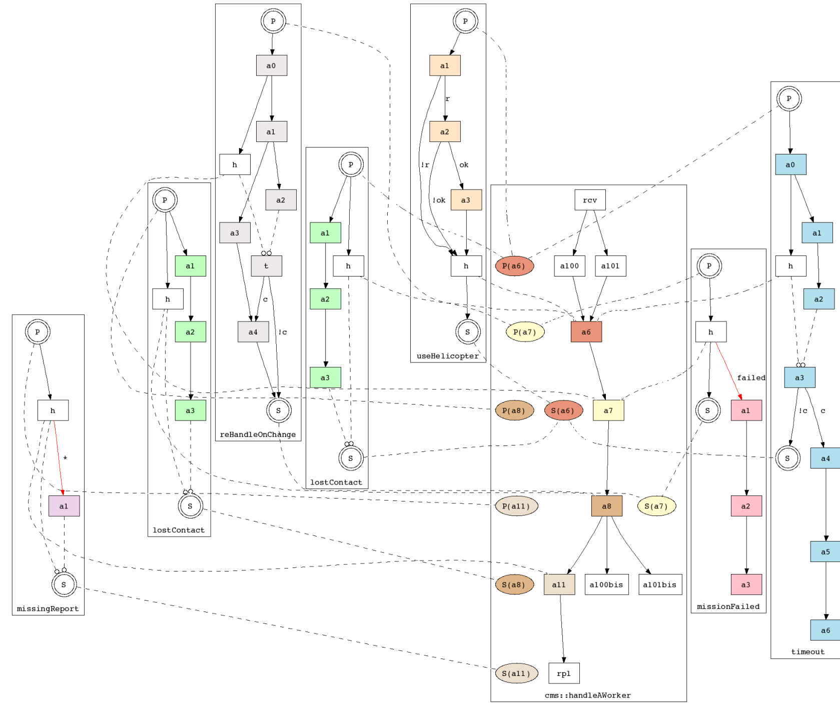

Fig. 1. Detailed visualization of a composition (at the activity level).

Such a detailed and focused view of composition is not scalable to support the design of large system. Visualization techniques of large data set such as

[3] To define this visualization, we extract raw information from the Adore engine and use a model transformation to obtain a Graphviz source code associated to these information. The source is then compiled using the `dot` tool to produce a Png file.

fish–eye [10] can tame the readability problems, but do not reduce the amount of details visualized in this representation. When designing a complete set of business processes using a compositional approach, the objectives of designers are to retrieve a holistic representation of the involved entities to understand easily what they are doing. Such a global visualization is needed at both design and analysis time.

Design Phase. When building a complete system using a compositional approach, designer needs to understand at a coarse–grained level the interactions between the different composition entities they are manipulating. At this step, designers focus on the fragments in terms of *impact* (*e.g.*, "the fragment throws a fault") instead of their detailed behavior (*e.g.*, "when a resource takes too much time to reach the crisis location, a timeout fault must be thrown").

Analysis Phase. After the composition directives execution, designers obtain a *composed* system. At this step, they need to identify critical points of the composed system, for example to design unit tests. This step focuses on the comparison between the *intrinsic complexity* of the original entities and the composed result.

3 Visualizing Compositions using Mondrian

We describe in this section the polymetrics view techniques, used to define three different visualizations of compositions. These visualization are then described and applied to the Cccms example.

The common objective of these three visualizations is to provide *dashboards* to designers. Composition dashboards are graphical representations meant to help designers to (*i*) get a scalable and global overview of the compositions present in an orchestration-based application, (*ii*) identity abnormal composition and (*iii*) facilitate the comprehension of a large composition by categorizing compounds. The idea of these dashboards is to enable a better comparison of elements constituting a program structure and behavior.

3.1 Polymetric views technique description

The visualizations we propose are based on the *polymetric view* [11]. A *polymetric view* is a lightweight software visualization technique enriched with software metrics information. It has been successfully used to provide "software maps" intended to help software comprehension and visualization. Figure 2 illustrates the principle of polymetric view.

Given two-dimensional nodes representing entities, we can map up to 5 metrics on the node characteristics: position properties X and Y, height property, width property and color property:

- *Size.* The width and height of a node can render two measurements. We follow the intuitive notion that the wider and the higher the node, the bigger the measurements its size is telling.

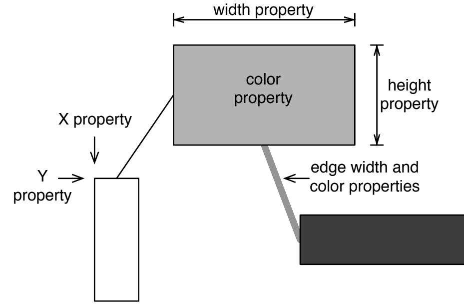

Fig. 2. Principle of polymetric view.

- *Color*. The color interval between white and black may render one measurement. The convention that is usually adopted [12] is that the higher the measurement the darker the node is. Thus light gray represents a smaller metric measurement than dark gray.
- *Position*. The X and Y coordinates of the position of a node may reflect two other measurements.

3.2 Fragments Dashboard *(Design Phase)*

This visualization represents all the *fragments* (as square) involved in a system, focusing on their *impact*. We have identified several impact properties represented as *boxes*: *(i)* hooked variable modification, *(ii)* exception throw, *(iii)* fault handling, *(iv)* initial process execution inhibition and finally *(v)* restricted process inhibition. To illustrate this visualization, we instantiated it on the Cc-cms example (Fig. 3). It represents the 24 fragments defined to answer to the different use–cases extensions defined in the requirements.
Interpretation. Based on the graphical representation obtained in this view, we have identified 7 different fragment categories, grouped into 3 main families: business extensions ($\mathcal{B}$), fault handling ($\mathcal{F}$) and control–flow inhibition ($\mathcal{I}$).

- Business extensions ($\mathcal{B}$): The *white* fragments only enrich existing process with new additional behavior. They do not modify the initial logic of the business process, and only add new features to enrich it.
- Fault handler ($\mathcal{F}$): *Yellow* boxes represent *fault handling* property. There are several ways to deal with a fault when it occurs in a process: *(i)* doing a re–throw (*green* property) to customize the fault, *(ii)* bypassing the fault (by modifying data to handle the problem, *blue* property) locally and *(iii)* handling the fault by using a business-driven reaction (no other property).
- Control–flow inhibition ($\mathcal{I}$): The *red* property represents the inhibition of an activity and its followers in a business process. The *pink* property is a restriction of the *red* one, since the fragment only inhibits the followers of

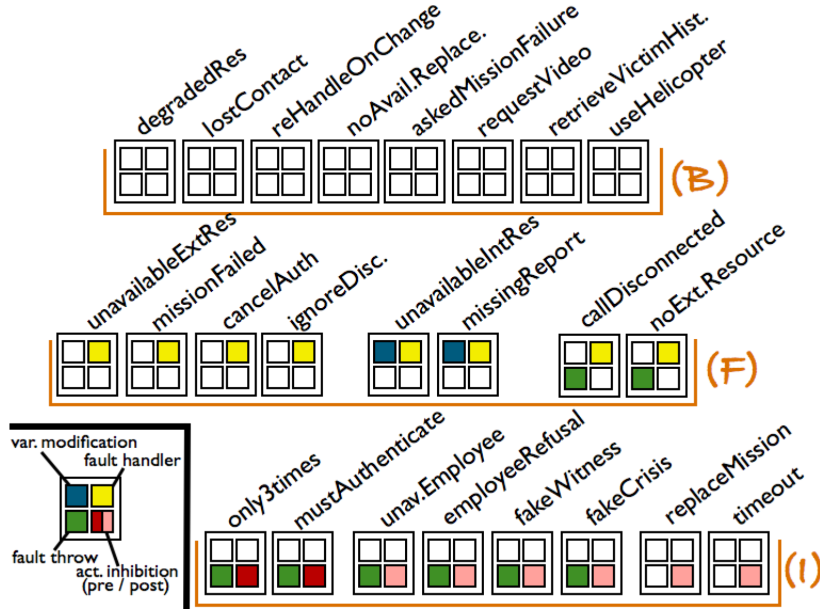

Fig. 3. Fragments dashboard instantiated on the CCCMS example.

an activity. Correlated with the fault thrower property (*green* color), we can identify *precondition* (activity inhibition & throw) and *postcondition* (followers inhibition & throw) checker. The *pink–only* fragments represents "dangerous" fragments, which inhibit several activities using a non–standard behavior. A deep understanding of the business–domain is necessary to grasp their behavior properly. The `timeout` fragment is a typical example of this kind of fragments: instead of "simply" throwing an exception when the system detect that a resource takes too much time to reach the crisis location, the CCCMS business model asks to stop whatever the system was doing and urgently inform a human coordinator able to decide what to do to counterbalance the situation.

Consequently, this view helps designers to perform a coarse–grained identification of their critical fragments. We have shown this criticality by explaining the *pink–only* fragments in the previous paragraph. Other dangerous color scheme are *green–only* (error throwing in parallel with legacy activities) and *red–only* (control–flow inhibition with a business–driven reaction).

3.3 Composition Dashboard *(Design Phase)*

The previously defined view helps designer to understand the different fragments used in a system. We focus now on the visualization of composition defined be-

tween fragments and business processes. The composition dashboard visualization represents business processes as rectangles, and fragment using the *4 squares* pattern previously defined. A link between two entities means that they are used together in a composition. The CCCMS case study instance of this visualization is depicted in Fig. 4. It represents the 24 fragments and their application on the 11 orchestrations used to realize the use cases.

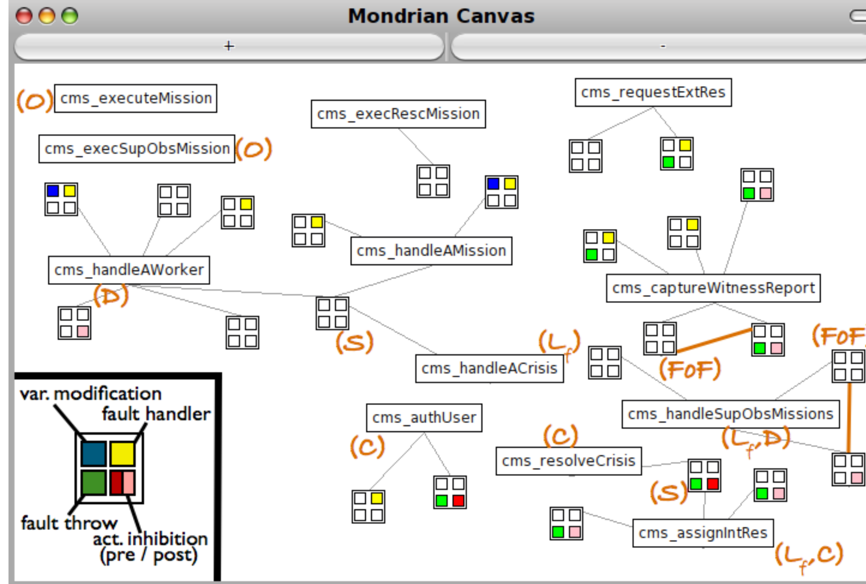

Fig. 4. Composition dashboard instantiated on the CCCMS example.

Interpretation. This visualization helps the designer to understand the depicted system in a holistic way. We identify the following composition categories:

- Orphans ($\mathcal{O}$): Orphans orchestrations are never involved in a composition (*e.g.*, `executeMission` and `execSupObMission`). Their identification helps to detect forgotten composition directives in a complex system. In the CCCMS context, they realize very basic use cases which do not define any scenario extensions. As a consequence, there is no fragment associated to these entities.
- Lack of Fault–handlers ($\mathcal{L}_f$): *yellow*–tagged fragments represent *fault handlers*. This visualization allows designers to easily identify a composition which does not involve any fault handler. The lack of fault handling in a process may leads to uncaught fault and jeopardize the behavior of the global system. It can also detects very basic processes which cannot fail.
- Conditioned behaviors ($\mathcal{C}$): Several fragments realize precondition (*red* & *green*) and postcondition (*pink* & *green*) checkers. Processes involved in a

composition which relies on such checkers must be considered as a good candidate for integration test definition.

- Cross-cutting / Shared concern ($\mathcal{S}$): We can easily identify two shared fragments in this case study. The first one is a precondition checker ("is the user authenticated?"), and the second one is a business driven preoccupation ("rehandle the crisis due to a change of external circumstances"). They clearly represent preoccupations which cross-cuts several scenario. Their identification helps to identify cross–cutting concerns and can drive system testing and re–engineering.
- Critical / Dangerous behavior ($\mathcal{D}$): Thanks to the *fragments dashboard*, the *pink–only* fragments were identified as *critical* (with other color scheme). Such fragments can inhibit the execution of a process subpart without using usual mechanism (*e.g.* fault throwing) to counter–balance their inhibition. As a consequence, orchestrations enriched using these fragments will require a specific attention from the designer.
- Fragment composed on other fragments ($\mathcal{F}o\mathcal{F}$): We can notice that several fragments are also composed with other fragments. This view lets designer identify such compositions and helps to grasp a semantic link between fragments when discovering an unknown system.

This view lets us clearly identify business–driven processes (using *white–only* fragments) from technical ones. Technical processes deal with precondition and postcondition checking, and do not involve any business driven fragment. On the contrary, high–level business processes rely on business–driven fragments to change the initial control–flow. It produces a dichotomy between *technical* processes (`assignIntRes`, `authUser`, `resolveCrisis`) and *business–driven* processes (the others). Based on this categorization, we can notice two things in the context of the Cccms:

- Categorizing the `resolveCrisis` process as technical seems weird as it corresponds to the main use case of the Cccms. But the scenario realized through this process is clearly technical (*i.e.*, opening a crisis case, asking partners to handle this crisis, closing the case when the crisis is ended) and does not involve any business–driven logic (realized in the other processes, which really *handle* the crisis).
- We can notice the `captureWitnessReport` process, which involves fragments dealing with postcondition checking, fault–handling and business–driven extension in the same composition. As a consequence, this process is both *technical* and *business–driven.* This process is critical in terms of Cccms business–domain: it realizes the retrieval of car crash witness information in a report and drives the trigger of specific missions according to this report. As a consequence, it does not fulfill a single objective and use very different fragments to achieve its complete behavior. This process can be then considered as an self–determining subsystem in the Cccms context.

Moreover, this visualization supports designers when debugging their compositions. This view can identify in a user–friendly way forgotten composition (through orphans processes), or the lack of fault handler or checker.

3.4 Composition Zoom *(Design Phase)*

When multiple fragments are composed with a process, designers need to restrict their visualization on the system to focus on a given composition. This *zoomed* visualization opens the orchestration box, describing where the fragments are integrated into the initial process. To emphasize the scalability[4] of this visualization, we use the `handleAWorker` process, which is the biggest composition in this case study. The obtained instance is represented in Fig. 5.

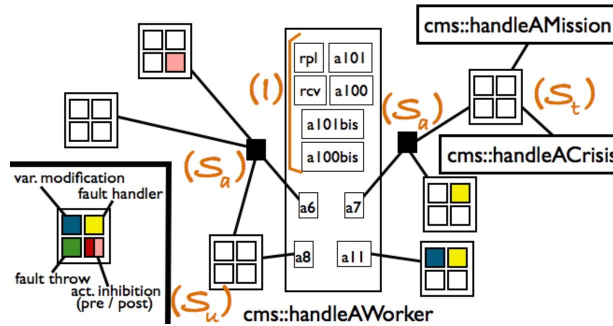

Fig. 5. Composition zoom instantiated on the `handleAWorker` process.

Interpretation. This visualization supports a focus on a specific composition and facilitates navigations throughout the orchestration set. Based on this representation, we can easily identify the following concerns in a given composition:

– Isolated activities ($\mathcal{I}$): some activities are not involved in any composition. They represent *technical* activities which do not interfere with the realized scenario. In this particular example, they implement initial message reception, final response sending and other technical activities
– Shared activities ($\mathcal{S}_a$): this visualization lets us clearly identify when several fragments are applied at the same location in a business process (*i.e.* a_6 and a_7 activities). Adore supports the automatic composition of these fragments into a merged one. However one may need to visualize them simultaneously to understand the merged behavior. In this example, fragments applied on a_6 deal with the non–arrival of a resource at the crisis location. Focusing only on these 3 fragments helps to support the design of the system: are all the non–arrival reasons handled by the process? Are these fragments semantically conflicting?
– Shared Fragments: this view lets the designer identify shared (*i.e.* used several times) fragments for a given composition. Such fragments usually represent cross-cutting concerns. The fact that a fragment is shared with other artifacts gives extra–information when working on its composition:

[4] The same composition is depicted at the activity level in Fig. 1.

- Multiple / Shared targets ($\mathcal{S}_t$): this fragment is related to others processes. This information is a fine–grained version of the "shared concern" one (from *composition dashboard*), linking a process activity to others processes through the use of a common fragment. It can drive the exploration of an unknown system by following such links from a process to another one.
- Multiple / Shared usage ($\mathcal{S}_u$): this fragment is a factorized enrichment of the initial scenario. As a consequence, it represents a chronic situation which requires special attention when testing and debugging the system. When a fragment is merged several time in the same process, it can introduce redundant activities in the final composition result. This view helps to understand the origin of such redundancy.

3.5 Complexity Dashboard *(Analysis Phase)*

When building a system using a compositional approach, the fragments added into the legacy business process enrich the initial behavior. It is interesting to visualize the difference between the initial and composed process to identify composition families. We use several indicators to model business process complexity (inspired by the ones defined by Vanderfesten *et. al*, [13]). In this view, we use the following indicators to represent the system. The complete CCCMS system is represented in Fig. 6.

- *width*: The *width* of a process represents the maximum number of activities executed concurrently in the control–flow.
- *height*: The *height* of a process represents the maximal length of the process.
- *maze*: The *maze* of a process represents the number of different paths available in the process. We map the *maze* indicator to the *color* dimension of the polymetric view.

Interpretation. Using this visualization, designer identify *composition families*, in terms of process complexity evolution (denoted as Δ_i, where i is a business process indicator).

- Δ_w (width expansion): When the composed process is larger than the initial one, it implies that several activities are executed in parallel of the initial behavior. Such an intensive parallelism is *resource-consuming*, and may be deployed on specific high–performance server. As a consequence, designers can identify from this visualization processes requiring a specific attention about performances after composition. In this example, the `handleSupObsMissions` process is a typical case of Δ_w. Fragments used in this composition introduce several notifications and interactions with other systems (*e.g.,* national crisis management center, internal message bus) done concurrently to the initial scenario.
- Δ_h (height expansion): The augmentation of a process height induces an execution control–flow longer than the initial one. The composition impacts the process execution time and the quality of service is then identified easily

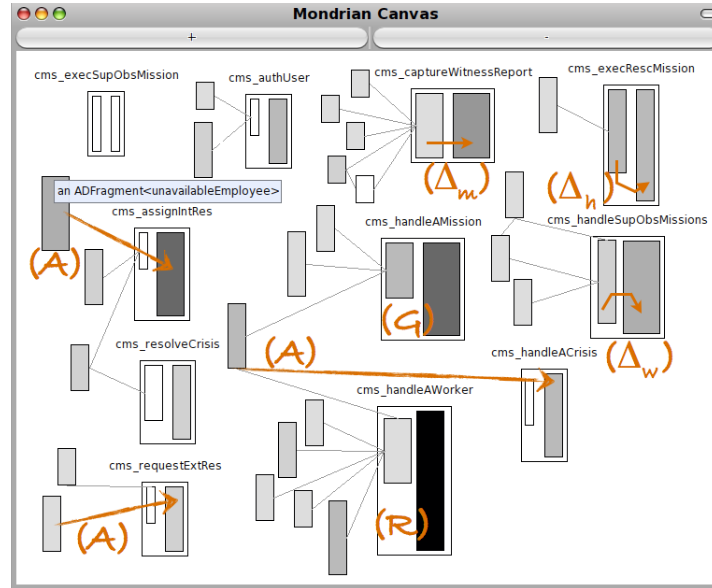

Fig. 6. Complexity dashboard instantiated on the CCCMS example.

using this visualization. In the `execRescMission` composition, the fragment introduces an interaction with an external system (retrieving victim's medical history before starting the health-care process).

– Δ_m (maze expansion): A dark color identifies a complex process, defining a lot of different paths in its control flow. A contrast change between an initial process and its composed result indicates that the activities defined in this process are more connected between each others. In the CCCMS example, the `captureWitnessReport` process is enriched with several small fragments. According to their *impact properties*, these fragments deal with condition checking and fault handling. As a consequence, the composed process contains a lot of new paths (*e.g.* fault bypass) available during the execution of the process. One should pay attention to the possible semantic interactions introduced by such a composition (*e.g* the process handles two faults f and f' but does not define what to do if these two faults happens at the same time).

Based on the previously explained Δ_i expansions, we identify three *critical* situations easily identifiable in this visualization:

– Global Expansion ($\mathcal{G}$): This phenomenon is identified by an expansion of all the Δ_i indicators. The `handleAMission` process illustrates this global expansion. This strongly indicates that the process designer should particularly focus on this process when designing test, as its apparent initial simplicity hides a very complex process once composed.

– Initial Process Absorption ($\mathcal{A}$): A process is *absorbed* by an extension when the composition output looks like the absorber in terms of complexity. Semantically, it often highlights a requirement granularity problem, where the behavior defined as scenario extensions is more complex than the initial scenario. The `requestExtRes` process illustrates this phenomenon, as it is absorbed by the *degradedRes* fragment. In this case, the system initially requires an external resource, and the extension defines all the actions to be performed when such a resource is not fully available to handle this particular crisis.
– Resonant Composition ($\mathcal{R}$): Resonance is a particular case of the Δ_m expansion. It indicates a lot of interactions between activities in fragments and process, resulting into a labyrinthine process after composition. Designers should handle these processes by taking care of their inherent complexity, focusing on test design and condition checking. The `handleAWorker` process is a typical example of such a resonance.

4 Implementation & Validation

Composition engine implementation. The concrete ADORE engine (process representation & composition algorithm) is implemented as a set of logical rules, using the PROLOG language. To make ADORE interoperable with other tools, we provide an export mechanism, based on XML. ADORE internal representation of orchestrations and business indicators can be exported as XML documents.

Visualization engine implementation. Our composition dashboards are rendered using MONDRIAN[5], an agile visualization engine. For the purpose of the experiment, MONDRIAN operates directly on a metamodel that reifies all the notions introduced in this paper. [6]

Validation. The ADORE framework was used in five different case studies[7], from a simple *proof of concept* to real-life systems. Business domain handled in these case studies are very diversified (*e.g.* integer arithmetic, web 2.0 folksonomies, information broadcasting inside academic institution). Visualization techniques presented here were applied with success to these five examples. We present the CCCMS example in this paper because we think it is the more pertinent to describe the approach (important set of processes leads to a scalability challenge).

5 Discussions & Perspectives

The visualization techniques described in this paper really helps the design of a very large system when using a compositional approach. Considering compositions as first class entities, we provide to designer a framework able to support

[5] `http://www.moosetechnology.org/tools/mondrian`
[6] `www.moosetechnology.org/tools/adore`
[7] Details here: `http://www.adore-design.org/doku/examples/start`

their design process. Based on these techniques, we identify chronic fragment patterns and sketch a categorization of these entities. Moreover, when analyzing the composed result, we identify critical points where process extensions interact violently with initial behavior. These critical points are easily identifiable using our graphical representation, and may leads to design weaknesses detection. Assuming the fact that the design fits the described requirements, such points can then highlight client requirements weaknesses.

In this paper, we voluntarily focus our visualization work on composition definitions. Our goal is to "understand" a system defined by composition, and support the designer during the design process. We never addressed performances visualization. In an SOA realized using orchestrations of Web Services, partnerships between services and processes is a key point for performance measurement. The invocation of an external partner costs a lot (in terms of data exchanged over the network). Even if ADORE proposes a simple visualization of process partners, a MONDRIAN visualization of the global *choreography* of services [2] will help the designer to easily identify bottlenecks and dangerous patterns in the designed system.

Defining a software using composition techniques often leads to conflicting situations [14,15]. ADORE defines a set of conflict detection rules to identify conflicts in the composed processes. For the same reasons that ADORE raw visualizations do not scale large system representations (too detailed data), designers retrieve from the framework a lot of details concerning the different conflicts detected by the application of detection rules over their models. The *composition zoom* visualization helps designers to identify interaction niche (*e.g.* shared activities), and then tend to reduce the scope of informations handled by the designer. Even if conflict detection mechanisms can be used to automate the detection of conflict, *pragmatic* conflicts[8] will always need an intervention of the designer to be handled properly. consequently the definition of a MONDRIAN visualization to support conflict resolution will support the global approach and helps designer during the composition process.

6 Related Work

Pfeiffer and Gurd address the problematic of aspect visualization [16]. They use *Treemaps* to provide a very abstract visualization of aspect oriented programs, based on a hierarchical organization of visualized entities (*e.g.*, package, class). We propose in this work an intermediate representation, focused on fragment (advice) semantic and independent of any hierarchical relation between entities. Moreover, the use of polymetrics views let us map process metrics to visualized entities.

Network monitoring community define tools such as NAGIOS [17] to supervise large networks. These tools define interactive visualization to monitor active

[8] which are induced by the business domain (*e.g.*, fault exclusion, overlapped conditions)

networks and collect incidents. Techniques used to facilitate the visualization of very large network can be reused to make our work more scalable.

All the visualizations presented in this paper are polymetric views. This visualization mechanism has been essentially used to assess software source code [18]. Using polymetric views to assess a software process models has not been considered so far. A number of work may be related however.

Visualizing dependencies instead software component instead of their composition has been a much more active research topic. For example, D'Ambros *et al.* [19] visualize how bugs are related to software components. A number of visualizations are providing, ranging from tree maps to complex graph-like structures, on which different layouts are applied.

A number of other visualizations are commonly employed to visualize software structure. Tree-maps [20] enables one to quickly relate the associated metrics, the counter balance is that it provides little help when relations between elements have their importance, as this is the case in our work.

Byelas and Telea [21] proposed a technique based on "splat texture" that identity *areas* in a software architecture. The idea is to fill a contoured area using this texture. This representation is efficient when dealing with scalability. Their approach is complementary to our, and may be well combined.

7 Conclusions

Composition mechanisms as such defined in the Adore framework help designers to build business process. Visualization techniques can then be used to graphically represent the compositions, and support designers when they assess their systems.

In this paper, we propose three visualizations intended to support designers' understanding of compositions. These representation allow designers to identify easily graphical patterns in their composition, and then identify key–points in composed systems. These key–points can be used in different way (*e.g.,* to define unit tests, to catch designers attention on a dangerous situation). We illustrate the scalability of the approach by visualizing the Cccms system. This large system was initially defined to be a comparison referential for Aom techniques. The different visualizations described in this paper are scalable enough to let us represent in an understandable way all the artifacts of this case study.

We focus our work on static visualization of compositions as first–class entities. An interesting perspective of this work is to define new representation dealing with composition conflict detection (*e.g.,* conflict highlighting) or business–process partnerships (*e.g.,* execution bottleneck).

References

1. MacKenzie, M., Laskey, K., McCabe, F., Brown, P., Metz, R.: Reference Model for Service Oriented Architecture 1.0. Technical Report wd-soa-rm-cd1, OASIS (February 2006)

2. Peltz, C.: Web Services Orchestration and Choreography. Computer **36**(10) (2003)
3. White, S.A.: Business Process Modeling Notation (BPMN). IBM. (May 2006)
4. OASIS: WS Business Process Exec. Lang. 2.0. Technical report, OASIS (2007)
5. Douence, R.: A Restricted Definition of AOP. In: European Interactive Workshop on Aspects in Software (EIWAS). (September 2004)
6. Liu, J., Batory, D., Lengauer, C.: Feature Oriented Refactoring of Legacy Applications. In: Int. Conf. on Soft. Engineering (ICSE), Shanghai, China (May 2006)
7. Kienzle, J., Guelfi, N., Mustafiz, S.: Crisis Management Systems, A Case Study for Aspect-Oriented Modeling. Requirements document for TAOSD special issue, McGuill University & University of Luxembourg (September 2009)
8. Mosser, S., Blay-Fornarino, M., Riveill, M.: Web Services Orchestration Evolution: A Merge Process For Behavioral Evolution. In: 2nd European Conference on Software Architecture (ECSA'08), Springer LNCS (September 2008)
9. Mosser, S., Blay-Fornarino, M., France, R.: Workflow Design using Fragment Composition (Crisis Management System Design through ADORE). Transactions on Aspect-Oriented Software Development (TAOSD) (2010) 1–34 submitted.
10. Sarkar, M., Brown, M.H.: Graphical Fisheye Views of Graphs. In: CHI'92: Proc. of the SIGCHI conf. on Human factors in computing sys., New York, NY, USA, ACM (1992) 83–91
11. Lanza, M., Ducasse, S.: Polymetric views—a lightweight visual approach to reverse engineering. Trans. on Soft. Engineering (TSE) **29**(9) (September 2003) 782–795
12. Gîrba, T., Lanza, M.: Visualizing and characterizing the evolution of class hierarchies. In: WOOR 2004 (5th ECOOP Wkshp on OO Reengineering). (2004)
13. Vanderfesten, I., Cardoso, J., Mendling, J., Reijers, H.A., Van Der Aalst, W.M.: Quality Metrics for Business Process Models. BPM and Workflow Handbook (2007) 179–190
14. Barais, O., Lawall, J., Le Meur, A.F., Duchien, L.: Safe Integration of New Concerns in a Software Architecture. In: 13th Annual IEEE International Conference on Engineering of Computer Based Systems (ECBS'06), Potsdam, Germany, IEEE (March 2006)
15. Szyperski, C.: Independently Extensible Systems – Software Engineering Potential and Challenges. In: Proceedings of the 19th Australian Computer Science Conference", Melbourne, Australia (1996)
16. Pfeiffer, J.H., Gurd, J.R.: Visualisation-based tool support for the development of aspect-oriented programs. In: AOSD '06: Proceedings of the 5th international conference on Aspect-oriented software development, New York, NY, USA, ACM (2006) 146–157
17. Barth, W.: Nagios: System and Network Monitoring. No Starch Press, San Francisco, CA, USA (2008)
18. Lanza, M., Marinescu, R.: Object-Oriented Metrics in Practice. Springer-Verlag (2006)
19. D'Ambros, M., Lanza, M.: Visual software evolution reconstruction. J. Softw. Maint. Evol. **21**(3) (2009) 217–232
20. Balzer, M., Deussen, O., Lewerentz, C.: Voronoi treemaps for the visualization of software metrics. In: SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization, New York, NY, USA, ACM (2005) 165–172
21. Byelas, H., Telea, A.C.: Visualization of areas of interest in software architecture diagrams. In: SoftVis '06: Proceedings of the 2006 ACM symposium on Software visualization, New York, NY, USA, ACM (2006) 105–114