Design for Adaptation of Distributed Service-Based Systems

Antonio Bucchiarone, Martina De Sanctis, Annapaola Marconi, Marco Pistore, and Paolo Traverso(✉)

Fondazione Bruno Kessler, Via Sommarive, 18, Trento, Italy
{bucchiarone,msanctis,marconi,pistore,traverso}@fbk.eu

Abstract. Internet of Services applications need to cope with a continuously changing environment, both in terms of the context in which they operate, and of the services, users and providers involved. In this setting, adaptivity is to be considered an intrinsic characteristic of applications rather than an exception to be handled. In this paper we propose a design for adaptation approach that fully exploits the advantages of the service-oriented paradigm to support the development and operation of service-based applications operating in highly dynamic environments. The approach is based on dynamic and incremental service composition and re-configuration techniques and is evaluated on a real-world scenario in the Smart Cities domain.

1 Introduction

The Internet of Services (IoS) foresees a future Internet in which the provisioning of, access to and use of services will be as widespread as content is today. The urgent need for a more efficient and sustainable society, together with the spread of ubiquitous communication networks, highly distributed wireless sensor technology, and intelligent management systems, makes the *Smart City ecosystem an ideal ground for IoS*. In this setting, the role of service-oriented computing is to enable the integration and interplay between new and legacy city services to support the creation and delivery of innovative and efficient services for the citizens [10].

A key challenge that needs to be overcome for this to become a reality, is the capability of dealing with the *continuously changing complex environment* in which Smart City applications operate. Consider for instance the case of a "smart children's mobility system", supporting service users (parents, children) and providers (drivers, teachers, traffic aids, volunteers) in their daily operation of children mobility services (e.g., school buses, walking buses, bike trains, ride-sharing among parents). The implementation of such a system requires to deal with a variety of heterogeneous services provided by autonomous entities (e.g., registration services provided by the school, volunteer managem ent, safety and traffic information from local police system, access to smart and wearable devices). In this setting, changes are not only frequent, they are an inner characteristic of the system. In particular, the system should be resilient to changes

A. Barros et al. (Eds.): ICSOC 2015, LNCS 9435, pp. 383–393, 2015.
DOI: 10.1007/978-3-662-48616-0_27

in existing services and in user requirements, as well as be open and extensible for new functionalities and facilities to become part of the system. This is made particularly challenging by the high degree of connection and interdependencies among system components which are provided by autonomous entities [1]. There is therefore the need to develop applications that are *adaptable "by design"* and to provide a *dynamic and scalable runtime environment* that makes them resilient to the aforementioned changes.

In this paper we propose a *design for adaptation approach* that supports the development, deployment and execution of service-based systems operating in dynamic environments and we discuss its effectiveness when applied to Smart cities applications like the "smart children's mobility system". The approach exploits advanced techniques for dynamic and incremental service composition [2], to effectively deal with changes occurring at different levels in the system. Each system component is modeled in terms of the functionalities it provides and of the conditions under which each functionality can be used (e.g. context/situation-related properties). At the same time, it is possible to partially specify the implementation logic of each component that is then automatically refined with (one or a composition of) functionalities provided by other components. This results in a dynamic network of components, where most changes can be handled at a local level, that is, affecting only those components directly related to the change and making the change transparent to the rest of the system.

The rest of the paper is structured as follows. Section 2 introduces the children mobility scenario used as a reference throughout the paper; Sect. 3 presents in details the proposed approach for the definition and operation of service-based systems that are adaptable by design; finally, Sect. 4 discusses the effectiveness and performances of the approach while Sect. 5 presents some related works and conclusions.

2 Motivating Scenario

Independent and active mobility has been proven to be fundamental for the development of children and adolescents. Plenty of initiatives have been set up in our cities supporting children's freedom of movement. Just to cite a few examples, we have school buses, walking buses (children walking together to school supervised by volunteer parents), children bike-trains (similar to walking buses, by bike), volunteers at crosswalks, ride-sharing among parents. However, if the aim is to deliver "smart children mobility services", these services cannot be managed each by itself, but they should become part of an integrated mobility solution, the *Smart Children Mobility System* (SCMS), that supports service users (parents, children) and providers (teachers, traffic aids and volunteers) in their daily operation and management of the different mobility services.

Figure 1 presents an overview of the different components in the SCMS. If we consider the part of the system related to the walking school bus (WSB), we notice that the system should support the registration of children, parents and

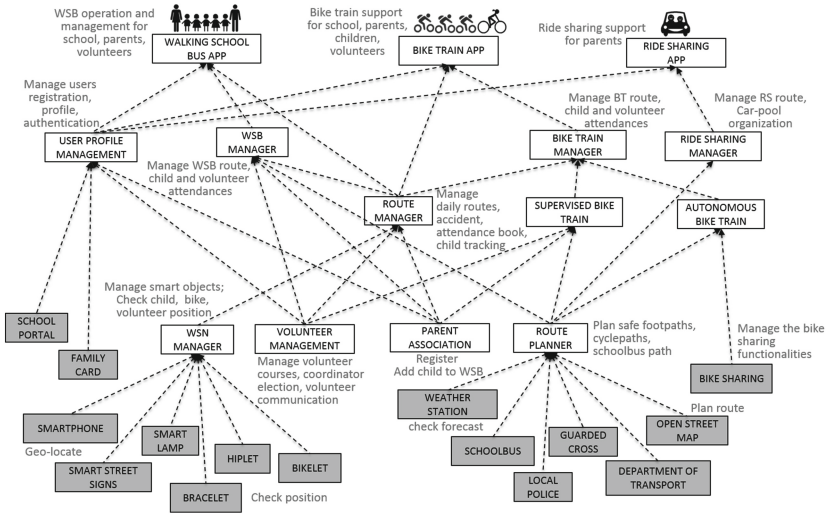

Fig. 1. Smart children mobility system: a partial overview of the system

volunteers and their access to the system (`User Profile Management` component), the training of volunteers (`Volunteer Management` component), the organization of routes (`WSB Manager` component) taking into account the needs of families but also route safety (e.g., presence of sidewalks, traffic situation, guarded crosswalks). The daily operation of the service requires to handle children attendance and volunteers availability for each route (`Route Manager`), the compilation of attendance books, tracking of children and volunteer position (`WSN Manager`), as well as managing possible exceptions (e.g. find a substitute for a volunteer, change the route due to roadworks, suspend the route due to weather conditions). Some components are in common with the bike train and ride sharing (e.g. `Route Manager`, `Volunteer Management`, `User Profile Management`). This allows not only replication avoidance, but also enabling synergies among the different services (e.g., exploit the ride-sharing service to cover a WSB route in case of bad weather).

A key characteristic of the system is the *variety and heterogeneity of services* involved: from domain-specific functionalities (e.g., management of walking bus routes, compilation of attendance books) to general-purpose ones (e.g., access management, user tracking); from back-end functionalities (in gray in the Figure) requiring the interaction with third party systems and devices (e.g. retrieving cycle lines, traffic/safety street information, interacting with smart objects) to front-end ones (e.g., Apps to be accessed by parents, volunteers, teachers). Moreover, the SCMS needs to deal with the *dynamicity* of the scenario, both in terms of the variability of services involved and of context changes affecting its operation. In particular, the system should be *open and extensible*, which means that new services (e.g., a new tracking device), as well as changes in existing

services (e.g., changes in parents authorization procedure, changes in any third party system) should require minimum maintenance. This is made particularly challenging by the collective nature of the services to be provisioned, since their operation requires the collaboration of different autonomous actors (school, parents associations, transport departments, local police), and it results in a *high degree of connection and interdependencies* among system components (as shown in Fig. 1).

3 General Framework and Approach

In this section we present the proposed approach for modeling and executing distributed adaptive service-based systems, such as the *SCMS* described in Sect. 2.

The system is modeled through a set of *domain objects* representing system components. Each domain object is characterized by a *core* process, implementing its own behavior, and a set of process *fragments*, representing the functionalities it provides (see Fig. 2). Fragments [5,11] are executable processes that communicate with the core process and that can be received and executed by other domain objects.

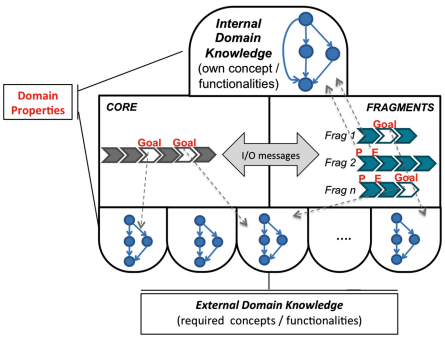

Fig. 2. Domain object model

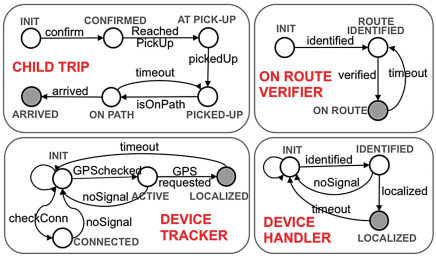

Fig. 3. Domain properties modeled as STS

Unlike traditional system specification, where components' behavior are completely specified, our approach allows the partial specification of the expected operation of domain objects through *abstract activities* that can be refined at run-time according to the fragments offered by the other domain objects in the system. For instance, the `Device manager` (see Fig. 4) can partially define the functionality for the localization of a device. Different smart devices can join the system and publish different tracking procedures. At run-time, when the device to be tracked is known (e.g. smart bracelet), the `Device manager` will use the fragments offered by the specific device to refine its abstract activity and to eventually locate the device. Abstract activities can be used both in the core process of a domain object as well as in the fragments it provides. The latter

case is more complex, and enables a higher level of dynamicity, since it allows a domain object to expose a partially specified fragment whose execution does not rely only on communications with its core process but also on fragments provided by other domain objects, thus enabling a "chain of refinements".

These dynamic features offered by the framework rely on a set of concepts, describing the operational environment, on which each domain object has a partial view. In particular (see Fig. 2), the *internal domain knowledge* captures the behavior of the domain concept implemented by the domain object, while the *external domain knowledge* represents domain concepts that are required to accomplish its behavior but for whose implementation it relies on other domain objects. Domain knowledge (both internal and external) is defined through *domain properties*, each giving a high-level representation of a domain concept (e.g. WSB route, child trip, device handler).

Consider for instance the domain property `Child trip` in Fig. 3 that models the typical daily trip of a child using a mobility service offered by the SCMS (e.g. walking bus). The participation of the child needs to be `CONFIRMED`, than the child reaches the pick up point (`AT PICK-UP POINT`) and she is `PICKED-UP` by a volunteer when she joins the ride. During the journey she might be `ON-PATH` (this is used to model the fact that at some times the system is certain of her position) and she eventually reaches the school (`ARRIVED`).

Each abstract activity is defined in terms of the *goal* it needs to achieve, expressed as domain properties' states to be reached. It is automatically refined at run-time, considering the set of fragments currently provided by other domain objects, the current domain knowledge configuration, and the goal to be reached. Activities in processes and fragments are annotated with *preconditions* and *effects*. Preconditions constrain the activity execution to specific domain knowledge configurations. For instance, in Fig. 4, the precondition `P2: Device tracker = active` says that, to execute the fragment `Geo-locate device` in the `Bracelet` domain object, the domain property `Device tracker` (see Fig. 3) must be in the state `active`. Effects model the expected impact of the activity execution on the domain and represent its evolution in terms of domain properties events. For instance, in Fig. 4, the effect `E2: On route verifier.verified` models the evolution of the `On route verifier` domain property (see Fig. 3).

Preconditions and effects are used to model how the execution of fragments is constrained by and evolves the domain knowledge. This information is used to identify the fragment (or composition of fragments) that can be used to refine an abstract activity in a specific domain knowledge configuration.

The resulting adaptive system is a dynamic network of domain objects. Potential dependencies (*soft dependencies*, from here on) are established between a domain object and all those domain objects in the system whose provided functionality (internal domain knowledge) matches with one of its required behaviors (domain property in its external domain knowledge). A soft dependency between two domain objects becomes a *strong dependency* if, during the system execution, they inter-operate by injecting and executing a fragment. The external domain knowledge of a domain object is not static since, if during a refinement

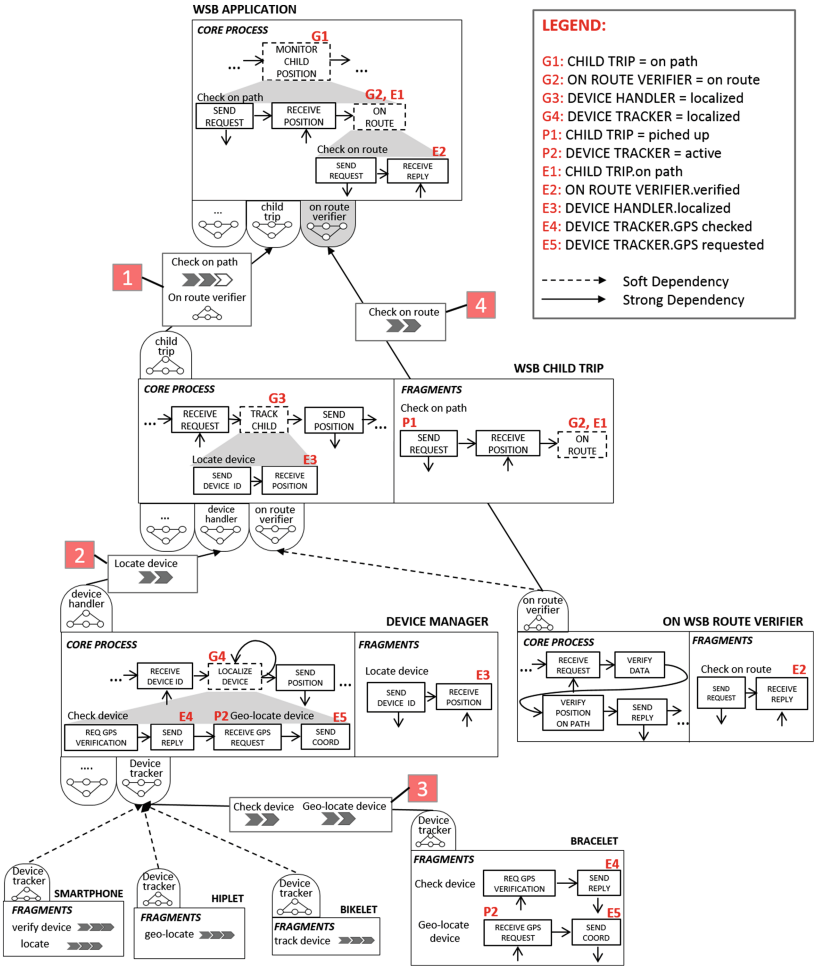

Fig. 4. A detailed example of dynamic refinements on the SCMS scenario

a domain object injects a fragment containing abstract activities in its own core process, it receives also the domain properties on which the fragment execution relies on, thus spanning its external knowledge. This dinamicity is reflected in the soft dependencies between domain objects because new dependencies might be established due to refinements.

In the following we present in details the different forms of refinement supported by the approach and how the adaptive system evolves at run-time. Consider, for instance, the scenario in Fig. 4 where a parent, or the school, wants to verify if a child is on the route of the WSB to which she has been subscribed.

Step 1. During the execution of the core `WSB application` process the abstract activity `monitor child position` needs to be refined. Its goal `G1: Child Trip =`

on path is defined over the Child Trip domain property (see Fig. 3). The refinement mechanism is triggered. The WSB child trip domain object implements the child trip domain property and exposes, among others, the fragment check on path. Supposing that this fragment is selected for the refinement, the first step of the refinement process consists in the injection of the check on path fragment in the behavior of the WSB application. Moreover, since the injected fragment contains an abstract activity, namely on route, the WSB application, together with the fragment, inherits the domain property on which the goal G2: On Route Verifier = on route is defined (Step 1 in Fig. 4). Thus, the WSB application domain object dynamically extends its knowledge boundaries and will use the received domain property to discover new domain objects (i.e., on WSB route verifier) and their related fragments. This step shows how fragments, together with domain knowledge, are exchanged by domain objects and injected at runtime in the core process. In particular, the WSB child trip domain object does not refine the on route abstract activity in its fragment, thus leaving the responsibility to refine it to the receiver domain objects.

Step 2. The check on path fragment, executed in the WSB application core process, communicates with the internal process of WSB child trip, which implements the functionality. During the execution of the WSB child trip process, the abstract activity track child needs to be executed. This activity has been defined as abstract since its execution depends on the tracking procedure supported by the device. The goal G3: Device Handler = localized on track child activity is expressed over Device handler domain property (see Fig. 3). The device handler behavior is offered by the Device manager through its fragment Locate device. A strong dependency is established between WSB child trip and Device manager, and the locate device fragment is injected in place of the track child activity (Step 2 in Fig. 4). This is an example in which, as opposite to Step 1, a domain object has an abstract activity in its core behavior, thus hiding to potential users of his fragments the fact that his behavior depends on third parties fragments.

Step 3. The execution of Locate device starts the execution of the Device manager's core process, which allows a device to be localized. The Device manager process contains an abstract activity, Localize device, which can be refined in different ways depending on the device to be tracked. This allows the Device manager to track any kind of smart object connected to the network, provided that it offers the possibility to be tracked. In our example, supposing that the child to be tracked has a bracelet, the Localize device activity is refined exploiting the functionalities offered by the Bracelet domain object. In this case, it offers the Check device fragment to verify the GPS connection's availability and, the Geo-locate device fragment to ask for the current position. Since the Geo-locate device fragment, needed to accomplish the goal G4, is constrained by the precondition P2: Device tracker = active, the Check device fragment is selected and composed with it as its execution, thanks to effect E4, satisfies the precondition P2. The obtained fragments' composition is then injected in place of the localize device activity (Step 3 in Fig. 4).

Step 4. The execution goes back to `Check on path` fragment in the core process of `WSB application`. Once child position has been obtained, last step consists in verifying if the child is on the route of the WSB. The `On route` abstract activity can be refined. The `WSB application` exploits the domain knowledge received by the `WSB Child Trip` in Step1, to find the fragments to refine the abstract activity. The `Check on route` fragment is selected and injected in the `WSB application` (Step 4 in Fig. 4) and the whole refinement process ends.

4 Evaluation

In this Section we evaluate the proposed solution both in terms of its effectiveness in defining adaptive systems that are resilient to a wide range of changes and in terms of the scalability of the automated composition techniques used for the refinement of abstract activities.

			ADAPTIVE SYSTEM MODEL		RUNNING INSTANCES	
TYPE	CHANGE	PROBABILITY	IMPACT	SUPPORTED SOLUTION	IMPACT	POSSIBLE SOLUTIONS
SYSTEM	Add component (internal/external)	High	Local	Add domain object, new soft dependencies	None	-
	Add provided functionality	High	Local	Add fragment in domain object, new soft dependencies	None	-
	Behavioral / structural change in provided functionality	Medium	Local	Change fragment specification (activities, preconditions, effects)	None / Local	Multiple process variants (None). Re-refinement (Local).
	Remove functionality	Low	Local	Remove fragment, reduction in soft dependencies	None -> Broad	Multiple process variants (None). Re-refinement chain (Broad).
	Remove component	Low	Local	Remove domain object, reduction in soft dependencies	None -> Broad	Multiple process variants (None). Re-refinement chain (Broad).
DOMAIN	New domain concept	Medium	Local	Add domain property and new domain objects/framents implementing it	None	-
	Change in a domain concept	Low	Broad	Update domain property and all related fragment annotations in domain objects	None -> Broad	Multiple process variants (None). Multiple re-refinement chains (Broad).

Fig. 5. Impact of system- and domain-level changes on an adaptive system instance

In Fig. 5 we list a set of changes typical of systems operating in the Internet of Services. For each change, together with its *probability* to occur, we present its impact on the adaptive system. We distinguish *system-level* and *domain-level* changes. Among system-level changes we have: the need to include new components (e.g. a new kind of wearable device in the SCMS), remove components, add/remove functionalities to existing components (e.g., the new version of the WSB Child Trip component allows changing pick-up point for a specific ride), as well as changing the operation (both in terms of behavior or data structures) of existing functionalities. With domain-level changes we mean changes in the domain concepts on which the dynamic operation of the adaptive system relies on (domain knowledge model).

When analyzing the resilience of the system to changes, we consider both the impact on the adaptive system model, that is, domain objects models on which future instances will be based, and on the running instances. In both cases we

present the impact of the change in terms of its scope (*local* to the affected component vs *broad*) and of the activities to be performed to apply the change to the system (*Solution*).

As emerges from Fig. 5, the resulting adaptive system model is resilient to all system-level changes (local impact). Moreover, the only change at domain-level having a broad impact is the case in which a domain concept needs to be revised. However, this latter change is the least likely to occur, since it implies a revision in the standard operation of a domain concept. This is usually related to changes in a business procedure or regulation, or when the way of doing something changes in a disruptive way (e.g., a completely new way of tracking a device).

If we consider the impact on running instances the situation is more complex. This is due to the fact that process instances, within domain objects instances, might be connected through strong dependencies, since some abstract activities have been refined with process fragments. For this reason, if the change concerns adding a new component or functionality, or a new domain concept to the system, then there is no impact at all on the running instances; whereas in some other cases the impact might span the instances subject of change and affect other instances in the system (broad impact). In Fig. 5 we suggest some *possible solutions* to limit the impact of the change or, in case this is not possible, to deal with it without affecting the system operation. A very effective solution, that makes every change transparent to running instances, is based on the combination of the proposed approach with techniques supporting multiple process variants (e.g. approaches based on software product lines such as the one in [9]). This would allow the definition of a new variant of the system according to the required change, on which all new instances will be based, and keep the old version of the system as a sibling variant on which running instances can continue their operation.

To conclude, we will briefly discuss the scalability of the automatic refinement techniques on which the approach is based. The key question is whether it is feasible to solve large number of refinement problems at run-time, each involving a potentially large set of domain objects and available fragments. The refinement mechanisms proposed in Sect. 3 have been implemented in the ASTRO-CAptEvo Framework [2] and tested on a real-world car logistic scenario (for a detailed description of the experiments please refer to [2]). The experiments consider over 2500 refinement problems on an adaptive system with 29 domain object types, 69 process fragment models and 40 domain properties models. On average, an adaptation problem contained 7 process fragments that could potentially be used in the final solution, and, as such, were presented in the planning domain. The experiments show that 90 % of problems were resolved within 0.1 seconds and 99 % within 10 seconds. These results, based on advanced optimization (problem reduction) and solution reuse techniques, clearly prove the feasibility of on-the-fly refinement for adaptive systems.

5 Related Works and Conclusions

In this paper we propose a design for adaptation approach that, exploiting advanced service refinement and re-configuration techniques, supports the design, development, and operation of service-based systems that are resilient to a wide range of changes. Various other approaches have been proposed in this direction. In the following we consider those approaches that might be applied in scenarios similar to the SCMS presented in this paper.

We will start by analyzing *rule-based approaches*. We mention MoDAR [12], a model-driven approach for the development of adaptive service-based systems. Rules are used to capture the variable part of a business process and linked in specific cutting points of the base process. In [8], the authors tackled the problem of the unpredictable execution of service-based applications by modeling composite services as artifacts that can change at runtime. Here, rules are used to model adaptation needs (events) and adaptation actions, from the design-time phase. However, rules are not suitable for managing continuous and unpredictable changes in *open* environments, since they require human intervention to be revised.

Another category of approaches, quite close to our proposal, that emerged in recent years are *artifact-centric approaches*. In [7] the authors present a formal framework defining *Business Artifacts* which represent conceptual entities made of their attributes and states, their tasks modeling services performed on such artifacts and business rules defined in ECA style specifying the lifecycle of an individual artifact, as well as the control logic of a process executing between interacting artifacts. Although the work in [7] supports flexibility and reusability, it suffers from the same limitations of rule-based approaches described in previous paragraph.

Other works [3,9] exploit the concepts of dynamic software product lines (DSPL) [6]. In DSPL a software family is analysed as a whole and both common and reusable assets are established, together with the possible customizations of the application. Feature models are used to specify alternative variations that can be used for adaptation. In [3] the authors bring forward the idea of their previous approach, called DAMASCo [4], which mainly consists in allowing services reuse in pervasive systems, and provide an extension based on both SOA and DSPL. Feature models are used to represent the variability of the services by modeling families of adaptable software products and to allow, as a consequence, the realization of dynamic service composition in a context-aware manner. In [9] the authors present LateVa, where similar processes containing common and variable parts are called process variants. Base process models are annotated with variation points where fragments can be dynamically entered. In this way, both software reuse and run-time variability are addressed. The main limitation of DSPL-based approaches is that they assume a close world where process variants (and thus fragments) are pre-defined. For this reason, these approaches do not fit well in open environments in which system components, and their provided functionalities, can enter or leave the system in any moment.

Although the approach proposed in this paper might overcome some limitations of existing works in the field, there are several open issues we would like

to deal with in the near future. Among them, the most important is an on-the-field evaluation of the approach experimenting the SCMS with end-users. A key extension is the possibility of verifying the current state of the external domain knowledge of each domain object through monitoring facilities offered by other domain objects (at the moment the state evolves considering only the effect annotations on the received fragments and might not be aligned with the real world situation). Another important extension concerns the support for other forms of run-time adaptation (e.g., reaction to a context change observed by monitoring the environment).

Acknowledgment. This work is partially funded by 7th Framework EU-FET project 600792 ALLOW Ensembles.

References

1. Bucchiarone, A., Cappiello, C., Di Nitto, E., Kazhamiakin, R., Mazza, V., Pistore, M.: Design for adaptation of service-based applications: main issues and requirements. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 467–476. Springer, Heidelberg (2010)
2. Bucchiarone, A., Marconi, A., Mezzina, C.A., Pistore, M., Raik, H.: On-the-Fly adaptation of dynamic service-based systems: incrementality, reduction and reuse. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 146–161. Springer, Heidelberg (2013)
3. Cubo, J., Gamez, N., Fuentes, L., Pimentel, E.: Composition and self-adaptation of service-based systems with feature models. In: Favaro, J., Morisio, M. (eds.) ICSR 2013. LNCS, vol. 7925, pp. 326–342. Springer, Heidelberg (2013)
4. Cubo, J., Pimentel, E.: DAMASCo: a framework for the automatic composition of component-based and service-oriented architectures. In: Crnkovic, I., Gruhn, V., Book, M. (eds.) ECSA 2011. LNCS, vol. 6903, pp. 388–404. Springer, Heidelberg (2011)
5. Eberle, H., Unger, T., Leymann, F.: Process fragments. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2009, Part I. LNCS, vol. 5870, pp. 398–405. Springer, Heidelberg (2009)
6. Hallsteinsen, S.O., Hinchey, M., Park, S., Schmid, K.: Dynamic software product lines. IEEE Comput. **41**(4), 93–95 (2008)
7. Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., Heath, F.T., Hobson, S., Linehan, M.H., Maradugu, S., Nigam, A., Sukaviriya, P.N., Vaculín, R.: Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In: DEBS 2011
8. Hussein, M., Han, J., Yu, Y., Colman, A.: Enabling runtime evolution of context-aware adaptive services (2013)
9. Murguzur, A., Trujillo, S., Truong, H.L., Dustdar, S., Ortiz, Ó., Sagardui, G.: Run-time variability for context-aware smart workflows. IEEE Software **32**, 52–60 (2015)
10. Pistore, M., Traverso, P., Paolucci, M., Wagner, M.: From software services to a future internet of services. In: Future Internet, Assembly, pp. 183–192 (2009)
11. Sirbu, A., Marconi, A., Pistore, M., Eberle, H., Leymann, F., Unger, T.: Dynamic composition of pervasive process fragments. In: ICWS, pp. 73–80. IEEE (2011)
12. Yu, J., Sheng, Q.Z., Swee, J.K.Y.: Model-driven development of adaptive service-based systems with aspects and rules. In: Chen, L., Triantafillou, P., Suel, T. (eds.) WISE 2010. LNCS, vol. 6488, pp. 548–563. Springer, Heidelberg (2010)