Mashup Development for Everybody

A Planning-Based Approach

Christian Kubczak[1], Tiziana Margaria[2], and Bernhard Steffen[1]

[1] Chair of Programming Systems, TU Dortmund, Germany
{christian.kubczak,steffen}@cs.tu-dortmund.de
[2] Chair Service and Software Engineering, Universität Potsdam, Germany
{margaria}@cs.uni-potsdam.de

Abstract. Today's service mashup technologies usually focus on assisting programmers to provide more powerful and valuable integrated applications to the users. A significant set of scripting languages, graphical tools and web services are used for this purpose, all addressing users with significant IT background. This paper aims at extending the power of mashup development to end users and application experts by automatically taking care of the tedious technical details like interface specifications, types, and syntactic constraints. In detail we support simple and intuitive mashup specifications which are automatically completed to runnable mashups by means of service discovery-like methods and planning. We illustrate our approach by means of a concrete case study executed within our jABC/jETI development and (remote) execution framework.

1 Introduction

When talking about mashups in the context of service oriented computing one usually means the combination of heterogeneous applications from different providers that yield completely new and more powerful applications that make extensive use of features of the reused services. Big internet companies and vendors meanwhile attract programmers by providing a developer-friendly support additionally to their pure web services. Examples are the Amazon Web Service Developer Connection [1] or APIs like the Google Web Tool Kit [2]. These technologies are easy to use for programmers at the code level, but inappropriate for domain experts as end users, who would like to combine their own special service mashup without being IT experts.
In this paper we aim at extending the power of mashup development to the end users and application experts by automatically taking care of the tedious technical details like interface specifications, types, and syntactic constraints. In detail we support simple and intuitive mashup specifications which are automatically completed to runnable mashups. In particular, this allows users to work with specifications which are loose in two dimensions:

– horizontally, i.e. concerning the orchestration of the involved services. This form of looseness, which is used to relax type matching and interfacing problems, is dealt with by means of planning along the lines proposed in [3–8] and
– vertically, i.e. concerning the refinement of specifications of individual services. This form of looseness is dealt with by means of service discovery technology and refinement [3–5, 9, 6–8, 10–13].

This user-centric top-level development fits within the eXtreme Model-Driven Design (XMDD) approach of [14]. XMDD is a new development paradigm designed to continuously involve the customer/application expert throughout the whole system's life cycle. In technical practice, user-level models are successively enriched and refined from the user perspective. Refinement stops at need, whenever a sufficient level of detail is reached, where elementary services that solve well defined tasks at the application level, can be implemented. The realization of these elementary services should typically be simple, they are often based on functionality provided by third-party and standard software systems. As the continuously enriched model is the central and sole artifact of this methodology, we also call this the *One-Thing Approach* [15, 16]. We illustrate here our approach by means of a concrete case study, focussing on the top-level development.

Sect. 2 briefly introduces jABC, our flexible service composition framework designed to support systematic development according to the XMDD paradigm, and its enhancement for remote service integration jETI [17]. Sect. 3 demonstrates our approach on the RichInfo Mashup case study and in Sect. 4 we enhance our manual modeling approach by introducing automated synthesis technologies. We then compare our work to related technologies and platforms (Sect. 5) and conclude (Sect. 6).

2 eXtreme Model-Driven Design with jABC

The jABC [18, 19] allows users to develop service-oriented systems by composing reusable building blocks (seen as services) into flow graph-like orchestrations called Service Logic Graphs (SLG). These Service Independent Building Blocks (SIBs) may represent single atomic service or also whole orchestrations (i.e., another SLG), thus SLGs can be hierarchical. This facilitates the refinement along the lines of the One-Thing Approach and grants high reusability not only of the building blocks, but also of entire models, as submodels within larger systems. Business rules and other correctness/compliance constraints can be defined for the SLGs, and they can be verified in jABC by formal methods like model checking. Finally, modeled SLGs can be compiled to a running system for a variety of target platforms [20].

SIBs arise typically from an available service description, like a WSDL or an API for REST, CORBA, or other library services, with different degrees of automation. They are grouped together by taxonomic descriptions, that are used for display and retrieval. If more information is available, e.g. as (semantic) annotations, we can use that too for service classification and discovery.

For the integration of remote services and tools into jABC we introduced jETI [17] as an enhancement framework. jETI provides a powerful but easy to use user interface to simplify both the integration and the use of remote resources and services. Different technologies are supported with different degrees of automation:

- **Web services** defined via WSDL documents [21] and handling communication via SOAP [22] can be automatically handled inside jABC.
- **REST)** services [23] in general provide no structured definitions. Their integration is done by using a guided import wizard which generates code stubs for the SIBs.
- **Proprietary 3rd party services** are integrated using customized GUI wizards integrated into the framework as needed. We support right now *SAP ERP* [24] services as well as statistical components of the *R-Project* [25].
- Additionally, jETI provides an own **lightweight remote service integration** approach, to support also occasional users *and* vendors with the "best of both worlds" of REST and standard Web services. Tools and services hosted on some remote server and described using an XML formatted document that is published.

How this works in detail has been described in [26]. Here, we illustrate how to use this technology for complex mashups, using as a case study the Rich Information provision service we demonstrated at the CeBIT 2009 in Hannover and at the IFA 2009 in Berlin.

3 Case Study: the RichInfo Service Mashup

The RichInfo service combines a variety of today's most used web services on the internet for the purpose of providing situated and consolidated information, pictures, references, and literature, to a person in a certain location. Variants thereof have been meanwhile tested as Assisted Living applications. This service, whose SLG is shown in Fig. 1,

1. locates and displays an address using Google Maps and OpenStreetMap (A and B).
2. tries to find nearby points of interest (POI) using GeoNames (C).
3. when a POI is selected, it looks for a related Wikipedia article, for pictures on Flickr, and searches for likely related products on Amazon (D and E).
4. additionally, it can proactively start communication actions with friends or peer located in the neighbourhood, using the completely web-service based NGN IMS Platform of Fraunhofer FOKUS in Berlin (not included here).

The 3 main parts of the service orchestration are outlined in Fig. 1. The central SIBs, that call the external services, are those highlighted with letters. They provide the typical functionalities an end user who wishes to describe the purpose of the service would be able and willing to mention. The other SIBs are

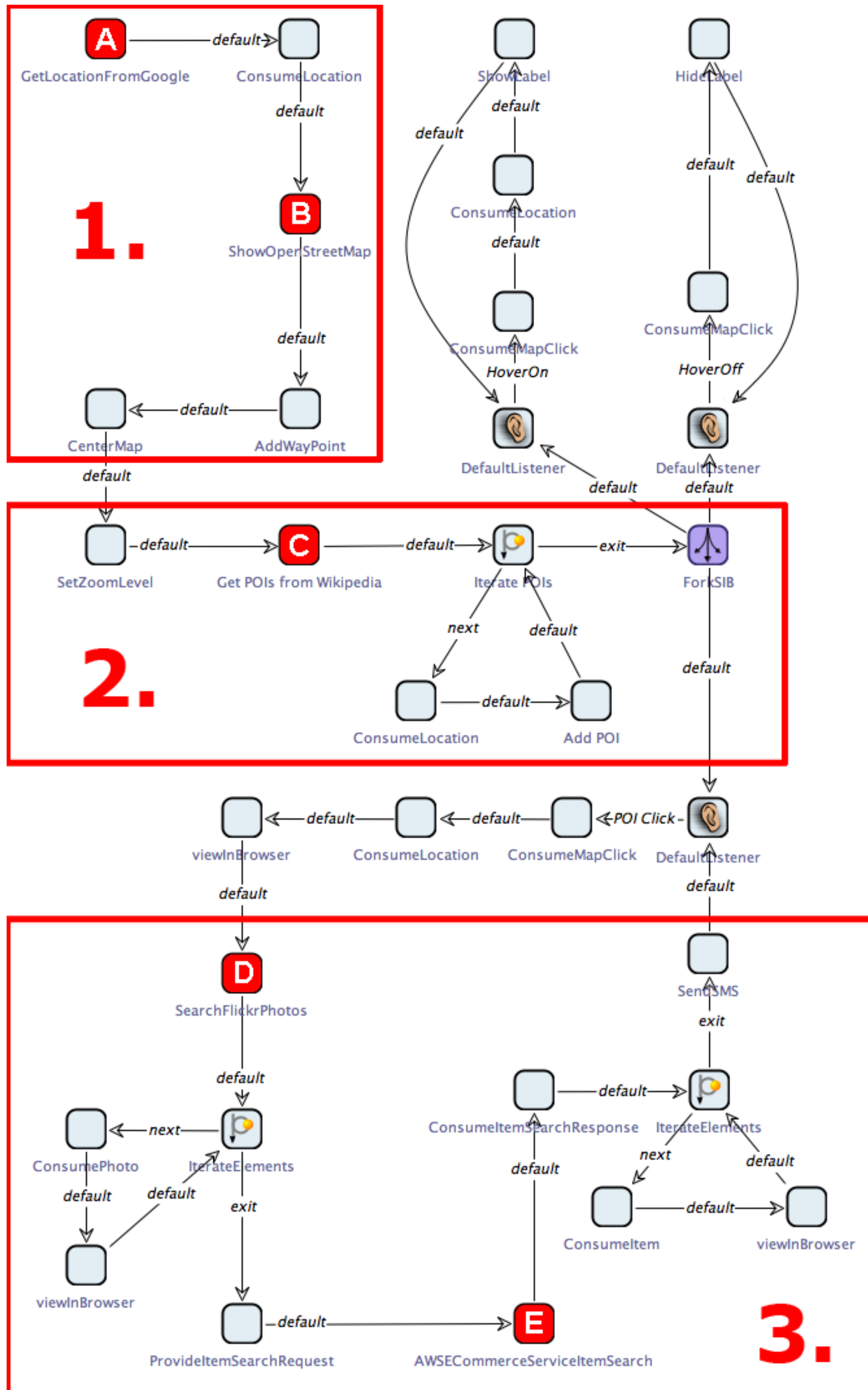

Fig. 1. RichInfo Mashup: using Google, OpenStreetMaps, Geonames, Wikipedia, Amazon and Flickr services

supporting components needed to deal with internal data of the workflow. To manually model the process, an application expert must either be aware of these and include them, or program a corresponding glue and adaption code from scratch. In Sect. 4 we show a synthesis approach where this need is eliminated, since services are orchestrated automatically regarding their functionality. In this service,

- a Google Map service (A) is called to return the geocode location (longitude and latitude) of a given physical address
- which is displayed inside an OpenStreetMap window (B) using a graphical waypoint.
- While the user now sees a detailed view of the specified address, a service provided by GeoNames (C) returns a given number of nearby interesting points and locations, that are added iteratively to the displayed map. The returned locations also contain information about related Wikipedia entries.
- The control flow is split by a Fork SIB into several parallel threads, that wait and listen to specified events that the user may trigger on the map. The two upper threads react to a mouse hover on/off a waypoint by displaying/hiding its name. The lower thread reacts to a mouse click on a waypoint by reading its returned location, opening a web browser, and displaying in distinct windows the related Wikipedia entry, Flickr photos (D), and Amazon articles (E).

This SLG can be further processed in the jABC: using the Genesys code generator, one can easily generate machine code for a mobile handheld, or generate and deploy it as a stand alone Web service.

Using an XMDD environment like jABC/jETI has several advantages over common technologies for mashup realization:

- even non-programmers can easily use this technology and graphically design their appropriate workflows
- once access to a (web) service is available as a SIB (whether directly implemented or imported), this service gains a high degree of immediate reusability by sharing the corresponding SIB library. Any other mashup can in fact reuse single features of the presented mashup by using that SIB, without touching code at all. As an example, inside this mashup the viewInBrowser service is reused several times
- agile evolution of the modeled application is intuitive, and actually native to the XMDD approach. As an example, considering the mouse event listeners of Fig. 1, Fig. 2(left) shows the three listener SIBs that react to mouse hover on/off and POI clicks. If one wishes to deactivate the mouse hover effect, one just needs to remove the corresponding edges, as in Fig. 1(right). Similarly, to change the flow of an application one can simply redirect a branch to the SIB or SLG of another service. For example, one could change this mashup to show information about a POI while hovering over a waypoint, instead of clicking it or deactivate the Amazon article lists.

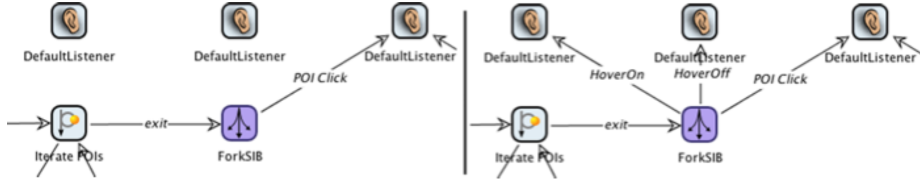

Fig. 2. Modifying the mouse hover reaction of the RichInfo Mashup: (left) before, (right) after

4 Intelligent Mashup Synthesis

Additionally to the modelling just shown, jABC also enables the user to automatically synthesize and verify service orchestrations, based on intuitive yet formal declarative specifications of *what* it should achieve. This way, anybody can focus on just describing the plain functional services of an application, even if incomplete. As we already mentioned, in the Rich Info mashup, of the entire SLG shown in Fig. 1 this concerns the services A to E. Ideally, one wishes that all the other services necessary to e.g. handle or (re)format internal data types could be ignored by the specifier (the real mashup designer) and be automatically added by the framework during synthesis.

In this XMDD perspective, instead of requiring a domain expert to be able to provide the whole set of components, we support an *application* expert, who describes a need by simply pointing out the core functionalities and thereby making use of an abstract view of the domain and of the orchestration. This significantly lowers the threshold of technical competence, and eases the task of providing powerful mashups. The following sections describe how we currently support this in the jABC.

We demonstrate here how to synthesize a complex mashup by applying a tableau-based software composition technique presented in [27, 4, 5]. It is an instance of LTL planning (or configuration) that refers to abstract semantic descriptions.

4.1 Abstract Semantics: Taxonomies for Categories and Types

The abstract information on the collection of basic services available for the mashup is shown in Fig. 3. The services are coarsely grouped in categories, here very simple, and each service is described by means of a name (here, the SIB name) and of abstract annotations concerning which entities they require and provide (input and output types, which are different from the programming-level types described in their WSDL or API, which is typically just string). In order to produce a running application these abstract SIB descriptions are then grounded to the concrete SIBs within jABC [28].

Simple ontologies (called taxonomies) express properties of the types (here, the SIB parameters) and services (here, the SIBs) of the mashup in terms of

category name	input type	output type	description
location			*location based services*
GetLocationFromGoogle	*searchStringIn*	*locationOut*	Gets a given address from Google
GetLocationFromInstaMapper	*{true}*	*locationOut*	Gets a location using InstaMapper
GetPOIsFromWikipedia	*latIn, lngIn*	*[true]*	Gets points of interest around the specified location using Wikipedia
helper			*helper services*
ShowInputDialog	*{true}*	*selectionOut*	Takes an input from the user
ConsumeLocation	*locationIn*	*titleOut*	Exracts information from a location
AddWayPoint	*mapIn*	*waypoointMapOut*	Adds a waypoint to a map
CenterMap	*waypointMapIn*	*centeredMapOut*	Centers a map
SetZoomLevel	*centeredMapIn*	*zoomedMapOut*	Zooms a map
viewer			*services used to view information*
ShowOpenStreetMap	*tagsIn*	*{true}*	Shows an Open Street Map at the given location

Fig. 3. The abstract service information for the main mashup services

has-a and *is-a relations* we call taxonomies. The taxonomies for our simplified RichInfo mashup are depicted in Fig. 4 and Fig. 5. This information serves as the concrete knowledge base which is used by the synthesis algorithm.

4.2 Declarative specification of the mashup: concrete and loose

The user we consider wishes to produce a mashup that, given a textual representation of an address, provides a map view of the corresponding location, together with points of interest in the surroundings, in other words

```
The user enters an address and gets a map showing the
location with nearby points of interests.
```

If we have the information of Fig. 4 available, the loose specification of the core services is simple: we need to query an address (SIB *GetLocationFromGoogle*), show it inside a map (SIB *ShowOpenStreetMap*) and find the nearby points of interest (SIB *GetPOIsFromWikipedia*). We may simply write:

```
(GetLocationFromGoogle < ShowOpenStreetMap < GetPOIsFromWikipedia)
```

using the services marked in Fig. 4, and using the symbol $<$ that means *before* or *precedes*.

The synthesis algorithm then generates one concrete solution that satisfies the requirement. This solution (see Fig. 6) contains only grounded services (SIBs), whose orchestration is guaranteed to be compatible with respect to the information on their input and output types.

However, the synthesis can take also a looser specification, containing taxonomic categories instead of concrete services and types. The loose specification language we support is Semantic Linear-time Temporal Logic (SLTL) [3],

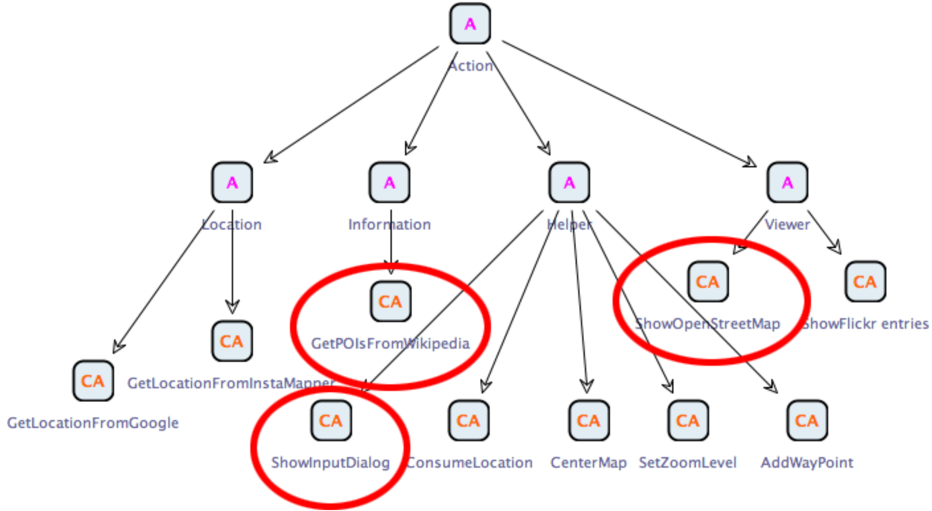

Fig. 4. The mashup's category taxonomy with the abstract actions (A) and the grounded SIBs (or concrete actions (CA))

a temporal (modal) logic comprising the taxonomic specifications of types and activities.

The concrete types *mapIn*, *centeredMapIn*, *waypointMapIn* and *zoomedMapIn* can be all seen as instances of an abstract type *map*, which decouples the rather technical layer dealing with data types on an implementation level from the domain specific business-level (Fig. 5). Similarly, instead of specifying concrete start- and end-services one could also be more generic, and wish to use a *location* service to get some kind of *view* and additional *information* of the surroundings:

```
(Location < Viewer < Information)
```

This makes it easy for an application expert to specify a rather loose wish, yet, due to the taxonomies and the grounding knowledge present in the system, it is already sufficient to obtain a concrete solution, as shown in Fig. 6. There we see that the synthesized solution directly corresponds to the first three service calls (A,B,C) in the original mashup of Fig. 1. Compared to the manual orchestration shown before, the application expert now only defines the (abstract) core service categories. The additional services used to provide supporting functionality are automatically added by the framework (here, as mediators that achieve type compatibility of the orchestration).

4.3 Declarative Refinement

The first solution generated by the system is not necessarily also the intended one. This is typically due to under- and sometimes also overspecification, that are dealt with in XMDD by means of agile change management and evolution.

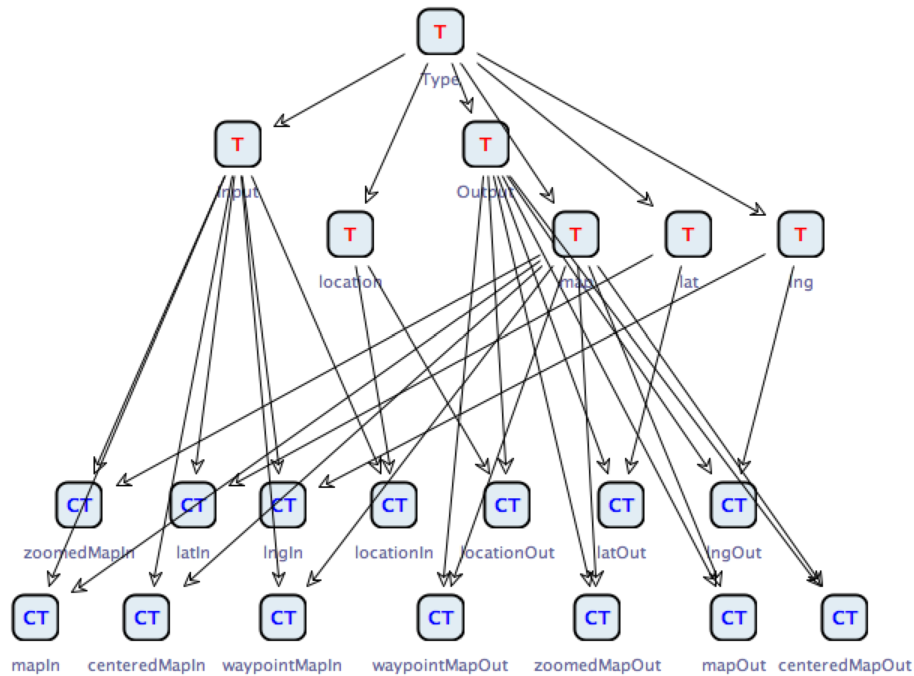

Fig. 5. The mashup's type taxonomy with the abstract types (T) and the bound SIB parameters (or concrete types (CT))

In our example, seeing the proposed solution, the user would perhaps rather prefer not to use the Google service, as its use in this specific context might be forbidden by Google's user license. SLTL properties can be used also to define such additional constraints on the synthesis, and this way refine the result. Via

```
not(GetLocationFromGoogle)
```

we force the algorithm to replace the service that gets the location. The result is shown in Fig. 7.

4.4 How to use the synthesis

The synthesis is steered from the jABC GUI. There, users can input the start/end services as well as additional SLTL formulas that describe the goal, and can ask for different kinds of solutions. The synthesis tool produces a graphical visualization of the satisfying plans (concrete service compositions), which can be executed or exported for later use. The synthesis algorithm takes as input a knowledge base consisting of the category and type taxonomies (Fig. 4 and Fig. 5) and of the service descriptions of Tab. 3. This knowledge base implicitly describes the set of all legal executions, called the *configuration universe*. This

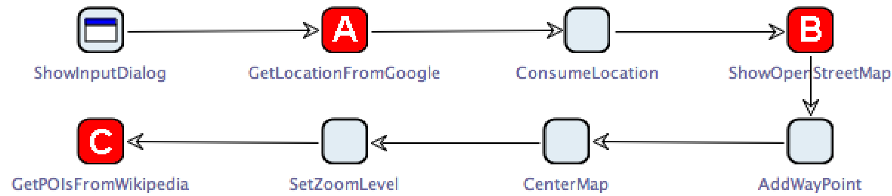

Fig. 6. The synthesized mashup solution

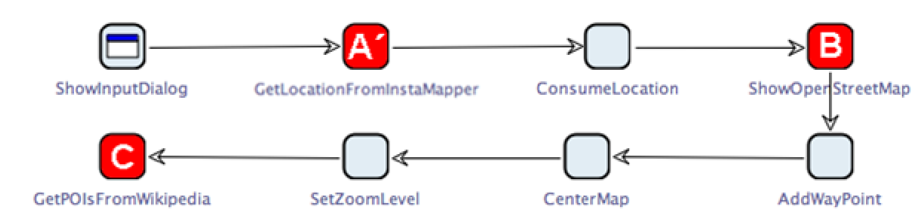

Fig. 7. The synthesized mashup solution avoiding Google services

universe could be very large as it contains all the compatible service compositions with respect to the given taxonomies and the collection of categories. Users are never confronted with the entire configuration universe. They see only the satisfying solutions, and they can choose which kind of solutions they would like to see:

- **minimal solutions** are plans (orchestrations) that achieve the goal without repetition of configurations. In particular, this excludes cycles;
- **shortest solutions** return the set of all minimal plans that are also shortest, measured in number of occurring steps;
- **one shortest solution** returns the first shortest plan satisfying the specification (as Fig. 6 of the above example);
- **all solutions** return all the satisfying solutions, including also cyclic ones.

The typical user interaction foresees a successive refinement of the declarative specification by starting with an initial, intuitive specification, that is often underconstrained, and asking for shortest or minimal solutions. Afterwards the graphical output is used for inspection and refinement (as done in Sect. 4.2 and Sect. 4.3).

5 Related Work

Many approaches try to lower the hurdles for dealing with (Web) service mashups. The Google Web Toolkit, based on Asynchronous JavaScript and XML (AJAX)

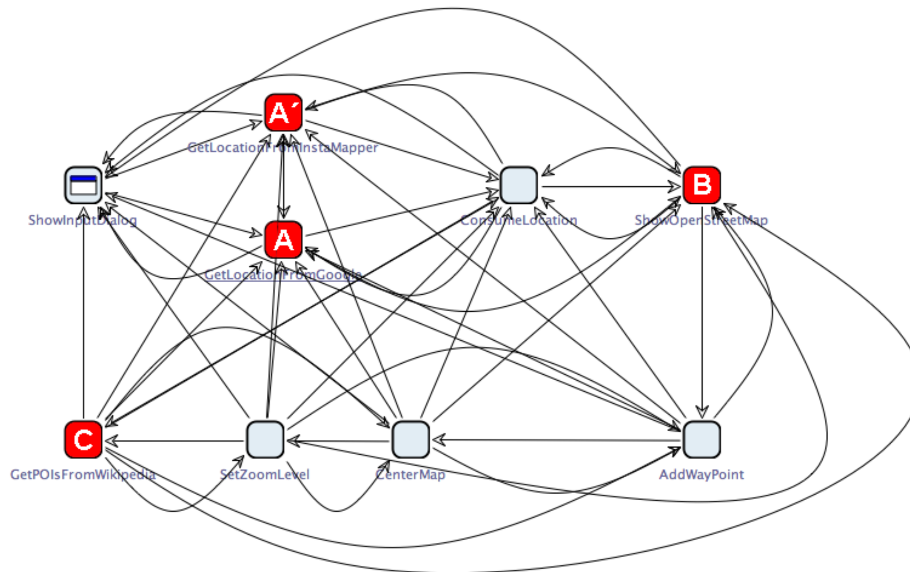

Fig. 8. The synthesized mashup solution with all valid paths

[29], was introduced to quickly set up applications based on various services provided by Google itself. Another popular and widely used framework is Ruby on Rails [30] which is based upon the Ruby object oriented programming language [31]. Like Java for the jABC, Ruby serves as the basic technology for the Rails framework. One of its goals is compactness, i.e. to minimize the lines of code a programmer must write to achieve full featured applications. The Rails framework extends this paradigm towards web based mashups by following the Don't Repeat Yourself (DRY) [32] and Convention over Configuration [33] principles, which support the programmer in agile software development. More similar to the jABC in the sense of semantics are frameworks introduced by Intel's Mashmaker [34], Yahoo Pipes, ORC [35] and Mashroom [36]: here, users are also able to focus on a mashup's data flow model and orchestrate given services. The compositions are realized using a minimal programming language containing basic operators inspired by Kleene algebra (ORC), by using an expressive data structure with a set of formally defined mashup operators based on nested tables (Mashroom) or by graphical, tree based representations of the data flow (Yahoo Pipes, Intel Mashmaker).

All these technologies still have one thing in common: a user has to bring at least some programming skills. Compared to jABC, Ruby On Rails, the Google Web Toolkit, AJAX or even plain Java programming using a web service framework like AXIS are more powerful when it comes to the degree of freedom a programmer has in the own hands. This holds as long as one excludes writing one's own SIBs, which is of course also possible in jABC and which then extends

the bounds to the power of plain Java. On the other hand, our approach is attractive for users not familiar with today's programming techniques and tools. This has been proven in several projects and courses involving non computer scientist, and even non technical personnel, eg. biologists now able to combine their own service mashups into agile and complex applications [37, 38]. The ORC programming language is somehow positioned between both extremes: it follows the philosophy of orchestrating services within a data flow structure, but it requires the user to bring at least basic programming skills.

6 Conclusion

We extended the power of mashup development to non-IT application experts by automatically taking care of the tedious technical details of correct compositions, like interface specifications, types, and syntactic constraints. In detail, we support simple and intuitive mashup specifications which are automatically completed to executable mashups by means of service discovery-like methods and planning techniques. We have illustrated our approach by means of a concrete case study implemented in our jABC/jETI framework. This RichInfo mashup is highly heterogeneous, and makes use of new technologies and platforms like Google's Android [39] or Apple's iPhone [40]. Since such platforms bring the idea of service mashups to consumer and communication hardware, they also claim the need for a mashup framework that is simple and easy to use for everybody. We believe that with the presented approach and technologies we can fill the gap between IT and business- or end users, thereby providing application experts a good framework to produce their own complex service orchestrations, and also covering business relevant domains like large scale enterprise systems, which can be dealt with in the same fashion, as demonstrated in the case of SAP's ERP products.
We are currently extending our work to a complex open framework, built to support profile-based best of breed technology. Concerning the underlying planning machinery, our framework already comprises, besides the tableaux-based synthesis algorithm with data flow types [5] used in this case study, four more approaches based on very different principles and techniques:

1. **situation calculus with Golog** [41], a tool that internally uses backward chaining for solving planning/synthesis tasks. This work is based on the well known situation calculus planner by Reiter and it is described in [42].
2. **monadic second order logic on strings, M2L(Str)** [43], which works by compositional automata construction in jMosel [44]. The solution is described in detail in [42].
3. a **tableaux-based synthesis algorithm with local input and output parameter types** where services are combined according to their compatibility using the inputs and outputs of their direct predecessor/successor as well as temporal logic constraints specified in Semantic Linear-time Temporal Logic (SLTL) [3, 27]. This solution is described in detail in [4].

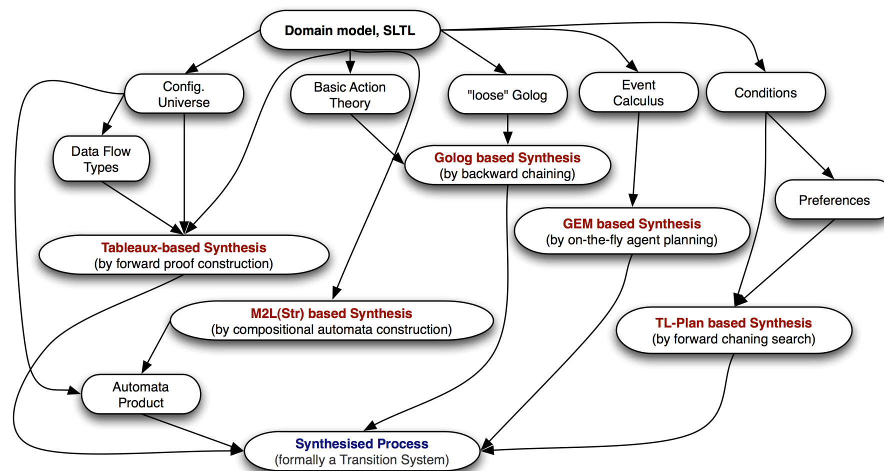

Fig. 9. The landscape of synthesis techniques in jABC

4. **Goal-oriented Enterprise Management (GEM)**, an abductive synthesis implemented in Prolog and based on [11, 12], that responds directly to situational changes using event calculus and agent techniques. This approach is explicitly described in [10].

Acknowledgment

We kindly would like to thank Stefan Naujokat for his work on the synthesis algorithm and its integration into jABC.

References

1. Amazon: Web Services. (2009) `http://aws.amazon.com/`.
2. Google: Web Tool Kit. (2009) `http://code.google.com/webtoolkit/`.
3. Freitag, B., Margaria, T., Steffen, B.: A pragmatic approach to software synthesis. In: Proc. of Workshop on Interface Definition Languages, ACM SIGPLAN Notices Volume 29 , Issue 8 (August 1994). (1994) 46–58
4. Margaria, T., Steffen, B.: LTL guided planning: Revisiting automatic tool composition in ETI. In: 31st Annual Software Engineering Workshop (SEW). (March 2007)
5. Margaria, T., Bakera, M., Kubczak, C., Naujokat, S., Steffen, B.: Automatic generation of the sws-challenge mediator with jABC/ABC. In: Semantic Web Services Challenge - Results from the First Year, Springer Verlag (2008) 119–138
6. Baier, J., Bacchus, F., McIlraith, S.: A heuristic search approach to planning with temporally extended preferences. In: Proc. of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07). (January 2007) 1808–1815

7. Baier, J., McIlraith, S.: Planning with temporally extended goals using heuristic search. In: Proc. ICAPS'06 Cumbria, UK,pp. 342-345, AAAI 2006, ISBN 978-1-57735-270-9
8. McIlraith, S., Son, T.: Adapting golog for composition of semantic web services. In: Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002), Toulouse, France (April 22-25 2002) 482–493
9. Kubczak, C., Margaria, T., Steffen, B., Winkler, C., Hungar, H.: An approach to discovery with miAamics and jABC. In: Semantic Web Services Challenge - Results from the First Year, Springer Verlag (2008) 199–234
10. Kubczak, C., Margaria, T., Kaiser, M., Lemcke, J., Knuth, B.: Abductive synthesis of the mediator scenario with jABC and GEM. In: Proc. 6th Int. Workshop EON-SWSC. Volume 359 of CEUR Workshop Proceedings. (June 1-2 2008) http://ceur-ws.org/Vol-359/Paper-5.pdf.
11. Kaiser, M.: Towards the realization of policy-oriented enterprise management. IEEE Computer **Special Issue on Service-Oriented Architecture**(11) (2007) 65–71
12. Kaiser, M., Lemcke, J.: Towards a framework for policy-oriented enterprise management. AAAI (2007)
13. Küster, U., König-Ries, B.: Semantic ssrvice discovery with DIANE service descriptions. In: Semantic Web Services Challenge - Results from the First Year, Springer Verlag (2008) 199–216
14. Kubczak, C., Jörges, S., Margaria, T., Steffen, B.: eXtreme model-driven design with jABC. In: Proc. of the Tools and Consultancy Track of ECMDA 2009. Volume WP09-12 of CTIT. (June 2009) 78–99
15. Margaria, T., Steffen, B.: Business process modelling in the jABC: The one-thing approach. In: Handbook of Research on Business Process Modeling, IGI Global (2009)
16. Margaria, T., Steffen, B.: Continuous model-driven engineering. IEEE Computer **42**(10) (2009) 94–97
17. Margaria, T., Nagel, R., Steffen, B.: jeti: A tool for remote tool integration. In: Proc. of TACAS. (2005) 557–562
18. Jörges, S., Kubczak, C., Nagel, R., Margaria, T., Steffen, B.: Model-driven development with the jABC. In: Proc. of HVC IBM Haifa Verification Conference. LNCS 4383, pp. 92-108, Springer Verlag (October 2006)
19. Margaria, T., Steffen, B.: Service engineering: Linking business and it. IEEE Computer **issue for the 60th anniversary of the Computer Society** (October 2006) 53–63
20. Jörges, S., Margaria, T., Steffen, B.: Genesys: service-oriented construction of property conform code generators. ISSE **4**(4) (2008) 361–384
21. W3 Consortium: Web Service Description Language. (2008) http://www.w3.org/TR/wsdl.
22. Web Services: SOAP. (2008) http://www.w3.org/TR/soap/.
23. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California (2000) http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
24. SAP: ERP. (2009) http://www.sap.com/ERP.
25. R-Project: Statistical Language. (2009) http://www.r-project.org.
26. Kubczak, C., Margaria, T., Steffen, B., Nagel, R.: Service-oriented mediation with jabc/jeti. In: Semantic Web Services Challenge - Results from the First Year, Springer Verlag (2008) 71–99

27. Steffen, B., Margaria, T., Braun, V.: The electronic tool integration platform: Concepts and design. Int. Journal on Software Tools for Technology Transfer (STTT) **2**(1) (1997) 9–30
28. Steffen, B., Narayan, P.: Full lifecycle support for end-to-end processes. IEEE Computer **11**(40) (November 2007) 64–73
29. Gustafson, A.: Asynchronous JavaScript and XML, Getting Started with Ajax. (2006) `http://www.alistapart.com/articles/gettingstartedwithajax/`.
30. Ruby: Rails Framework. (2009) `http://rubyonrails.org/`.
31. Ruby: Programming Language. (2009) `http://www.ruby-lang.org`.
32. Ruby: Don't Repeat yourself. (2008) `http://wiki.rubyonrails.org/rails/pages/DRY`.
33. Chen, N.: Convention over Configuration pattern. (2006) `http://softwareengineering.vazexqi.com/files/pattern.html`.
34. Ennals, R., Gay, D.: User-friendly functional programming for web mashups. In: Proc. of the 12th International Conference on Functional Programming (ICFP). (2007) 223–234
35. Misra, J.: A programming model for the orchestration of web services. In: Proc. of the Second International Conference on Software Engineering and Formal Methods (SEFM). (September 2004) 2–11
36. Wang, G., Yang, S., Han, Y.: Mashroom: end-user mashup programming using nested tables. In: WWW. (2009) 861–870
37. Kubczak, C., Margaria, T., Fritsch, A., Steffen, B.: Biological lc/ms preprocessing and analysis with jABC, jETI and xcms. In: Proc. ISoLA 2006, 2nd Int. Symp. on Leveraging Applications of Formal Methods, IEEE Computer Society (November 2006) 308–313
38. Margaria, T., Kubczak, C., Steffen, B.: Bio-jETI: a service integration, design, and provisioning platform for orchestrated bioinformatics processes. BioMed Central (BMC) Bioinformatics Supplement on Network Tools and Applications in Biology **9**(4) (2007) `http://www.biomedcentral.com/1471-2105/9?issue=S4`.
39. Google: Android. (2009) `http://www.android.com`.
40. Apple: iPhone. (2009) `http://www.apple.com/iphone`.
41. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
42. Margaria, T., Meyer, D., Kubczak, C., Isberner, M., Steffen, B.: Synthesizing semantic web service compositions with jMosel and Golog. In: Proc. ISWC 2009. (October 2009)
43. Church, A.: Logic, arithmetic and automata. In: Proc. Int. Congr. Math.,Uppsala 1963, pp. 23-35, Almqvist and Wiksells (1963)
44. Topnik, C., , Wilhelm, E., Margaria, T., Steffen, B.: jMosel: A stand-alone tool and jABC plugin for M2L(Str). In: Proc. of Model Checking Software 13th International SPIN Workshop. Volume 3925 of LNCS. (2006) 293–298