A Multipurpose Framework for Model-based Reuse-oriented Software Integration Synthesis

Alexander Perucci
University of L'Aquila, Italy
alexander.perucci@graduate.univaq.it

Marco Autili
University of L'Aquila, Italy
marco.autili@univaq.it

Massimo Tivoli
University of L'Aquila, Italy
massimo.tivoli@univaq.it

***Abstract*—Systems are increasingly built by reusing and integrating existing software. This paper presents the preliminary version of a multipurpose framework for software integration synthesis. The objective is to provide both researchers and practitioners with an easily accessible environment that, integrating different kinds of software synthesizers, permit to perform different kinds of analyses, verification, model-to-model and model-to-code transformations, all oriented to the reuse and the integration of existing, possibly third-party, software.**

I. Introduction

The Future Internet[1] (FI) promotes a large-scale computing environment that will be increasingly surrounded by software that, at different granularities, can be reused and composed to build value added services.

Systems are increasingly produced by integrating existing software. Furthermore, the focus of system development is on integration of third-parties software that exposes interfaces describing the provided functionalities and the way to interact with them, i.e., the behavioral interaction protocol.

Reuse-based software engineering is becoming the main development approach for building both business and commercial systems [1], [2]. Numerous methodologies and software engineering approaches have been proposed to address the problem of automatic synthesis of integration code and architectures, as well as integration paradigms and patterns [1], [3]–[8]. The proposed approaches encompasses a variety of formally grounded and practical aspects, ranging from specification and analysis, from model transformation to code generation, model checking and formal verification, from implementation to run-time management.

During the last decade, our research activity has been focused on model-based reuse-oriented software synthesis [2], [9]–[14] for integrating (centralized or distributed) software systems, from system architecture to composition code, from coordination to protocol mediation/adaptation. This research activity has been funded by a number of national and EU projects, among which the FP7 CONNECT[2], the FP7 CHOReOS and its follow-up H2020 CHOReVOLUTION[3], the GAUSS[4] and D-ASAP[5] Italian PRIN projects.

[1]www.future-internet.eu
[2]www.connect-forever.eu
[3]www.choreos.eu – www.chorevolution.eu
[4]www.lta.disco.unimib.it/GAUSS
[5]d-asap.ws.dei.polimi.it

This paper initiates a new research and development action that we intend to pursue towards realizing a multipurpose framework for software integration synthesis. The objective is to provide both researchers and practitioners with an easily accessible environment that, integrating different kinds of software synthesizers, permit to perform different kinds of analyses, verification, model-to-model and model-to-code transformations, all oriented to the reuse and the integration of existing, possibly third-party, software.

With this goal in mind, we present the MultiSynth STUDIO[6], a web application[7] that supports model-based synthesis of software integration code, as well as different kinds of analysis and verification. In its preliminary version, the studio provides support for realizing choreography-based systems by integrating existing software peers in a fully distributed way.

The MultiSynth STUDIO offers extension mechanisms so to include other kind of integration synthesis methodologies. That is, our mid-term goal is to include in the studio all the synthesis methodologies we have proposed in the past, so to offer a ready-to-use integrated working environment where practitioners can experiment with different synthesis methodologies, meeting both theoretical and practical interests.

The paper is structured as follows. Section II sketches a general picture of our multi-purpose integration synthesis methodology. Section III describes an instance of the general synthesis process to support the automated synthesis of choreography-based systems and its implementation. Section IV reviews related work, and Section V concludes the paper.

II. Model-based Synthesis overview

In its most general sense, a multi-purpose integration synthesis process might involve quite different activities and manipulated artifacts. Figure 1 shows a high-level activity diagrams describing the multi-purpose integration synthesis process that we have in mind.

The System Design activity concerns the definition of the models to be given as input to the synthesis process. Input models can be a mix of functional models, such as (global or peer-style) interaction protocols, and non functional ones, such as security requirements and QoS constraints. For instance, input models can be XML-based specifications of

[6]The code is freely available at https://github.com/sesygroup
[7]multisynthstudio.disim.univaq.it

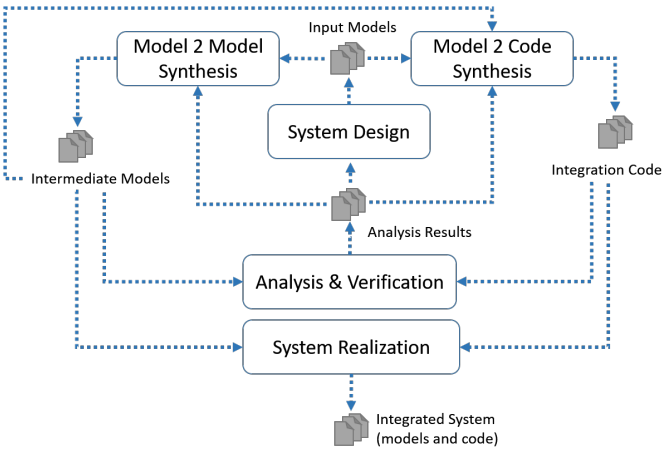

Fig. 1. Model-based Synthesis Approach

the interfaces (such as WSDL[8]) together with automata-based specifications of the interaction behavior of the third-party services/components being reused.

These models can then undergo both a Model-2-Model Synthesis step to be transformed into intermediate models, and/or a Model-2-Code Synthesis step to be transformed into actual code. Examples of output (intermediate) models can be the system architecture, coordination and adaptation models, deployment descriptor and, in general, any intermediate representation that supports further synthesis steps or analysis and verification. Examples of actual code can be additional software artifacts (implemented in any programming language) that, e.g., when integrated with the third-party software to be composed, contribute to built the overall final system while fulfilling the specified functional and non functional system properties.

Both intermediate models and code can be analyzed and verified to check, e.g., desired system properties, such as safety and liveness, QoS requirements fulfillment, structural properties of the input and intermediate models, and code, as well as the system realizability. The results of the Analysis and Verification activity can lead to reiterate the entire process, by possibly refining input models.

When the goal of the synthesis is to realize the final system, beyond analysis and verification, the System Realization activity may return an overall model, e.g., specifying all the possible interactions of the composed system, as well as, a bundle (e.g., a Web Archive or a JAR file) containing all the synthesized software artifacts (e.g., coordinators, adapters and security filters) together with a description of their dependencies for actual deployment and execution.

III. Choreography Synthesis

In this section we describe an instance of the general synthesis process to support the realization of choreography-based systems. Choreographies are an emergent service engineering approach to compose together and coordinate distributed software peers by describing the interactions among them from a global perspective. Choreographies model peer-to-peer communication by defining a multiparty protocol that, when put in place by the cooperating participant peers, allows for reaching the overall choreography goal in a fully distributed way.

Out of a choreography specification, the goal is to automatically generate a set of Coordination Delegates (CDs). CDs are additional software entities with respect to the choreography participants, and are synthesized in order to proxify and control the participants interaction. When interposed among the choreography participants, CDs enforce the collaboration prescribed by the choreography specification. The synthesized CDs are correct by construction, meaning that the resulting choreographed system realizes the choreography specification, even when not realizable by simply projecting the choreography specification into each single participant [2], [9], [11]–[13]. Our synthesis method deals with *hybrid* choreography participants that can communicate synchronously and/or asynchronously.

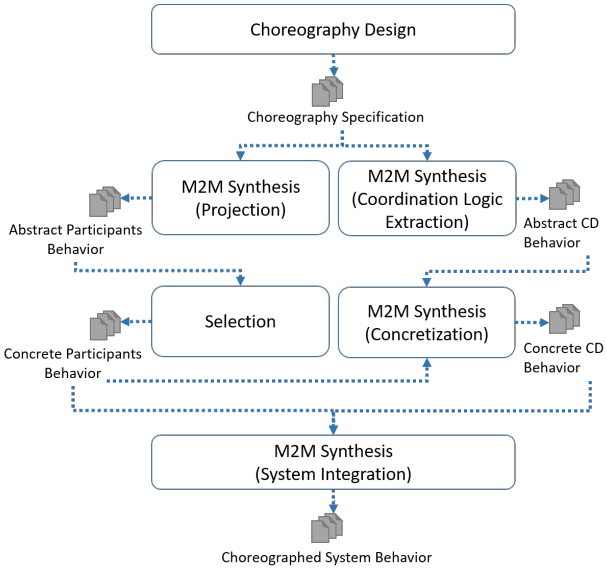

Fig. 2. Choreography Synthesis Approach

The choreography synthesis process supported by the MultiSynth Studio consists of the following six activities (Figure 2), each manipulating models conforming to aptly defined metamodels. Model conformance ensures that each model satisfies the constraints captured in its metamodel, i.e., that the model is indeed a valid instance of the metamodel [15].

Choreography Design – This activity concerns the definition of the choreography-based system to be realized. The choreography specification is given in terms of a state machine where a transition from a state to another models the exchange of a message between two peers. The choreography specification describes the way peers perform their interactions from a global perspective by focusing on the exchange of messages. Thus, a choreography specification defines the (partial) order

[8]https://www.w3.org/TR/wsdl

in which the message exchanges occur. Each message exchange involves two peers: the sender and the receiver of the message. The choreography specification abstracts from the way peers communicate to exchange messages, i.e., synchronous versus asynchronous communication. The communication style will be concretized after concrete participants (e.g., software services, software components, and things) are selected to play the roles of the choreography peers.

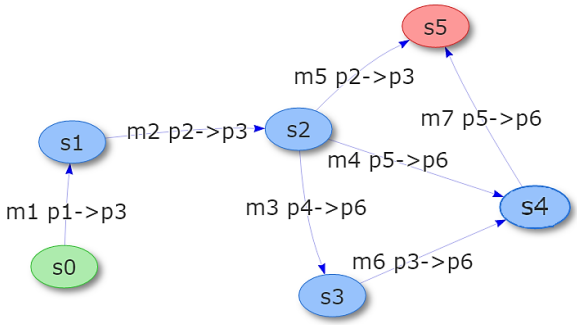

Fig. 3. Sample Choreography Specification

Figure 3 shows a sample choreography specification. It describes the interactions of the peers P1 to P6. The possible interactions are such that P3 receives the message m1 from P1, followed by P3 receiving the message m2 from P2. Then, when in the branching state s2, three exclusive alternatives can be undertaken. One alternative accounts for P6 receiving m3 from P4, followed by P6 receiving m6 from P3, in turn followed by P6 receiving m7 from P5. The other two alternatives account for either P6 receiving m4 from P5, or P3 receiving m5 from P2. Each possible alternative path leads to the state s5, from where no more outgoing transitions can be undertaken. The choreography specification conforms to the metamodel depicted in Figure 4. The metamodel consists of an arbitrary number of Messages, Participants, Transitions, and States. Exactly one of these states is an initial state. Each state serves as source and/or target for a transition. Each transition is concretized as SendingMessageActionTransition to model the exchange of a message between a source participant (i.e., the one sending the message) a target participant (i.e., the one receiving the message).

M2M Synthesis (Projection) – Out of the choreography specification, the projection step generates one behavioral model for each choreography peer. This model is a state machine where a transition represents the action of either sending or receiving a message (observable actions), or an internal action (not observable from outside). The transition labels postfixed with "!" represent send actions; the transition labels postfixed with "?" represent receive actions; the label "epsilon" indicates an internal action. For a given choreography peer, a projection represents the behavior expected by the concrete participant that will be selected to play the role of the choreography peer, according to the sequences of message exchanges specified by the choreography. Being derived from the choreography

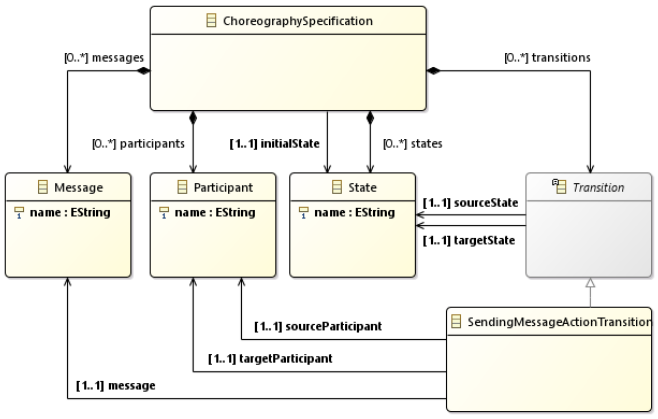

Fig. 4. Choreography Specification metamodel

specification, also this model abstracts from the type of the send and receive actions (synchronous or asynchronous). For this reason, we call this model Abstract Participant Behavior.

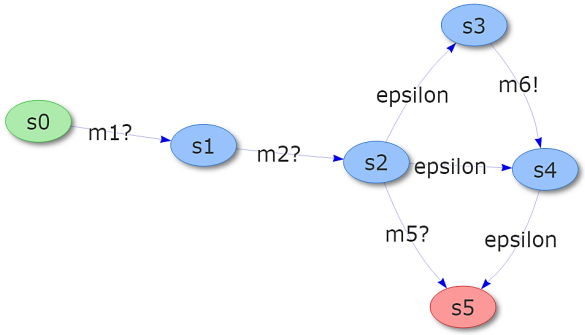

Fig. 5. Abstract Participant Behavior P3

Figure 5 shows the abstract participant behavior of the participant P3. For example, from the state s0 to s5, a possible expected execution is such that P3 receives the message m1, followed by the message m2. Then, when in the branching state s2, one alternative is that of receiving m5.

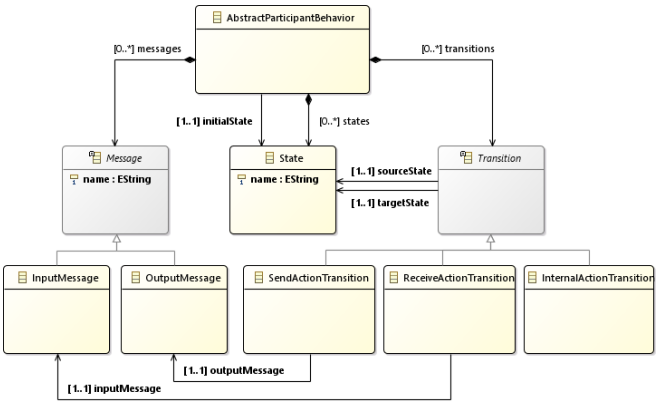

Fig. 6. Abstract Participant Behavior metamodel

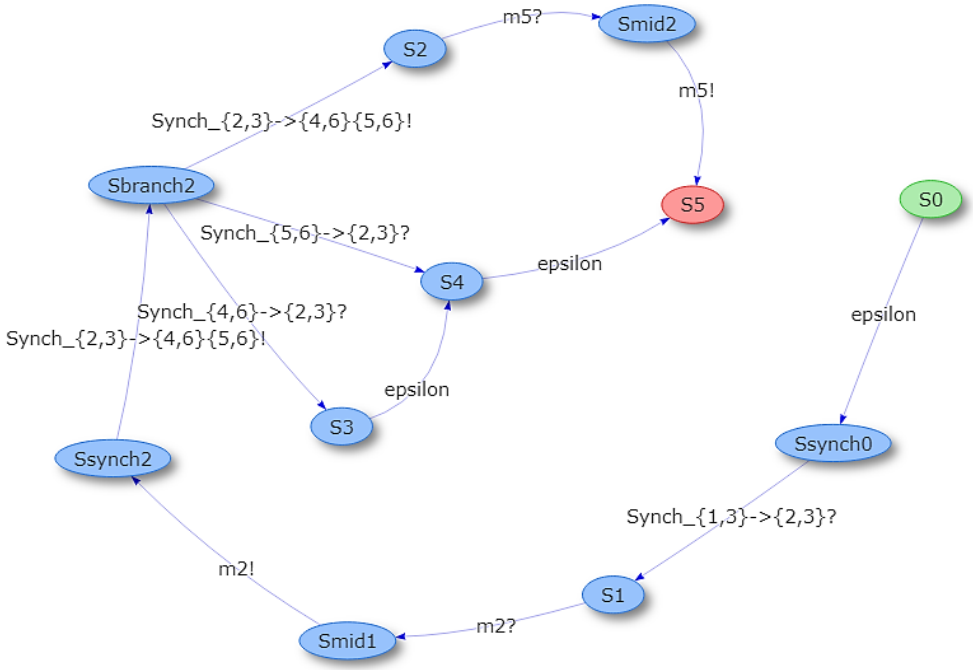

Fig. 7. Abstract CD Behavior for coordinating P2 and P3

Note that we have deliberately left internal actions, which indeed could be collapsed through an automaton simplification procedure. The related metamodel is shown in Figure 6. By analyzing the metamodel, we can notice that (differently from the choreography metamodel in Figure 4) a message is concretized by OutputMessage and InputMessage in order to distinguish between sent messages from received ones, respectively. A transition is concretized by SendActionTransition, ReceiveActionTransition, and InternalActionTransition.

M2M Synthesis (Coordination Logic Extraction) – This step takes as input the choreography specification and automatically extracts the coordination logic that is required to coordinate the choreography peers in a distributed way. The extracted coordination logic is thus distributed into a set of Abstract CD Behavior models. Similarly to the Abstract Participant Behavior, each of them is a state machine where a transition models the action of either sending a message or receiving a message, or an internal action. As better detailed later on this section, these actions are related to the *standard communication* performed to achieve the choreography business logic (see Figure 10). Differently from the Abstract Participant Behavior, there are also transitions modeling the synchronous exchange of coordination/synchronization messages. These actions model *additional communication* required to realize the coordination logic that is needed to enforce the realizability of the specified choreography. Standard communication takes place between a CD and the participant it controls and supervises. When needed, additional communication messages are exchanged among the involved CD.

Figure 7 shows the logic of $CD_{\{2,3\}}$ coordinating the choreography peers P2 and P3. The coordination logic is such that $CD_{\{2,3\}}$ waits for receiving a synchronization message from $CD_{\{1,3\}}$. Only after it can receive the message m2 from P2 and forward it to P3. When in the state "Ssynch2", $CD_{\{2,3\}}$ send a synchronization message to $CD_{\{4,6\}}$ and $CD_{\{5,6\}}$ reaching the branch state "Sbranch2". When in this state, three exclusive alternatives can be undertaken. From top to bottom, one alternative accounts for a send synchronization message to $CD_{\{4,6\}}$ and $CD_{\{5,6\}}$, followed by a send and then a receive of the message m5; the other two alternatives account for receiving synchronization from $CD_{\{5,6\}}$ or $CD_{\{4,6\}}$, respectively, in turn followed by internal actions. The coordination logic model also conforms to the Abstract Participant Behavior metamodel in Figure 6.

Selection – Since our approach is reuse-oriented, the choreography is not implemented from scratch; rather, the approach allows to enforce the realizability of choreographies that reuse, as much as possible, e.g., third-party services published in a service inventory. Then, the selection activity consists of selecting concrete participants capable of playing the roles of the choreography peers. This calls for exogenous coordination of the selected participants since, in general, we cannot access their code or change it.

The output of the selection phase is a set of the behavioral specifications, each one defining the interaction protocol of the selected participants. A behavioral specification is also an automata-based model that we call Concrete Participant Behavior. The related metamodel (not shown for lack of space) is similar to the metamodel in Figure 6, with the difference that for each transition, its type is specified: synchronous,

asynchronous, or internal.

Figure 8 shows the behavior of concrete participants selected for the choreography peers $P1$ to $P6$. The messages $m3$, $m4$, $m6$, $m7$ are exchanged synchronously (graphically represented as continuous arrows); $m1$, $m2$, $m5$ are exchanged asynchronously (graphically represented as dashed arrows).

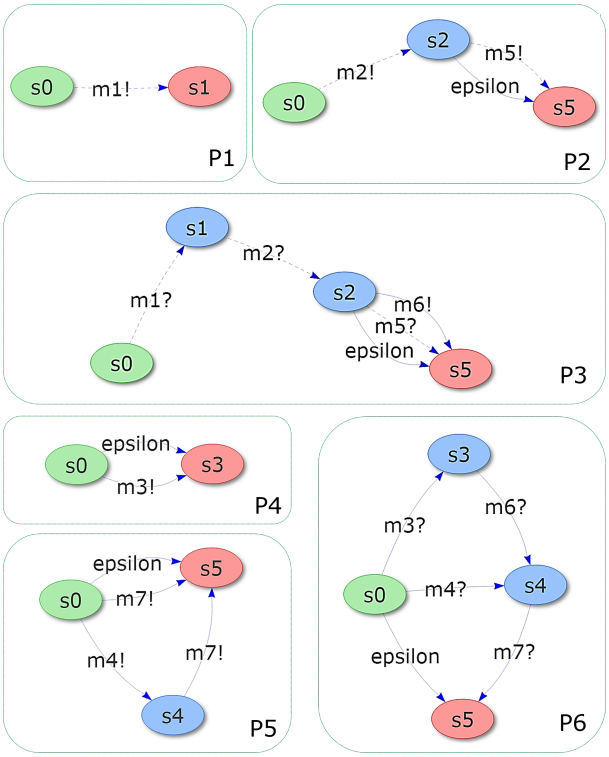

Fig. 8. Behavior of the Concrete Participants P1 to P6

Figure 9 shows an excerpt of the hybrid system (manually drawn) obtained by composing the set of concrete participants $P1$ to $P6$ in Figure 8 (the automatically generated model is not reported here since too large in size). In Figure 8, we have named the states so to facilitate the mapping of the choreography states in Figure 3 to the states of the hybrid system behavior resulting from the composition in Figure 9. For each participant Pi, its states are labeled as Pi*:state-name*.

In the hybrid system in Figure 9, asynchronous interactions ($\circ$) are handled through messages queues all having maximum size equals to 1, i.e., the participants $P1$, $P2$ and $P3$ are all assigned 1-bounded queues; whereas, the participant $P4$, $P5$ and $P6$ are all assigned 0-bounded queues – they exchange only synchronous messages ($\bullet$). The participant $P3$ is indeed a hybrid participant since it exchanges $m6$ synchronously, and $m1$, $m2$ and $m5$ asynchronously. In the figure, message queues are denoted as [...]. Initially, all the queues are empty, hence []. State changes and queue updates are highlighted in bold step by step.

Following the branch on the left-hand side of Figure 9, three different paths can be undertaken from the state marked with **(1)**. The corresponding traces are:

$m1_{P1\rightarrow P3} \in m2_{P2\rightarrow P3} \in m5_{P2\rightarrow P3} \in$
$m1_{P1\rightarrow P3} \in m2_{P2\rightarrow P3} \in m3_{P4\rightarrow P6}\ m6_{P3\rightarrow P6}\ m7_{P5\rightarrow P6}$
$m1_{P1\rightarrow P3} \in m2_{P2\rightarrow P3} \in m4_{P5\rightarrow P6}\ m7_{P5\rightarrow P6}$

It is easy to see that these three traces are all allowed by the choreography in Figure 3. Conversely, all the traces (not completely shown in Figure 9) traversing either the state marked with **(2)** or the state marked with **(3)** are not allowed by the choreography. Thus, according to the notion of choreography realizability given in [16], [17], the choreography in Figure 3 is not realizable. Intuitively, this basically means that if we let the selected participants interact in an uncontrolled way in the composed system, they can perform interactions that are not permitted by the choreography specification.

M2M Synthesis (Concretization) – Since the Abstract CD Behavior is a priori generated out of the choreography specification only, it abstracts from the way the selected participants communicate. This information is added by the Concretization step, just after the Selection step. Interested readers can play with the MULTISYNTH STUDIO web application to visualize the resulting concrete CDs.

M2M Synthesis (System Integration) – For the set of synthesized CDs, correctness by construction means that when they are composed with the selected participants (System Integration step), the behavior of the integrated choreographed system realizes the specified choreography. That is, the synthesized concrete CDs enforce by construction the realizability of the specified choreography. According to a predefined architectural style, CDs are interposed among the participants needing coordination. Figure 10 shows the instance of the architectural style related to the sample choreography in Figure 3.

CDs perform coordination (i.e., *additional communication* in the figure) of the participants interaction (i.e., *standard communication* in the figure) in a way that the resulting collaboration realizes the specified choreography. According to the type of actions performed by the concrete participants, standard communication can be synchronous or asynchronous. Additional communication is always synchronous. In this sense, the system behavior is modeled as a Hybrid System Behavior model. Interested readers can visualize the latter by using our web application. The related metamodel can be found under the github repository.

Analysis & Verification, M2C Synthesis – It is out of scope for this paper to discuss all the kinds of analysis and verification steps that can be performed on the generated models (beyond the fact that here we have described only the choreography-oriented synthesis process). Just to mention a few, concrete participants behavior that model third-party services and CDs can be analyzed and verified to check if the choreographed system can be effectively realized taking into account specified system properties. Reachability paths, deadlock freeness, dead loops, sink states, etc, can also be analyzed. The discussion of the different M2C transformations that can be employed is also outside the scope of this paper. Interested reader can however refer to our previous work in [2], [9]–[14].

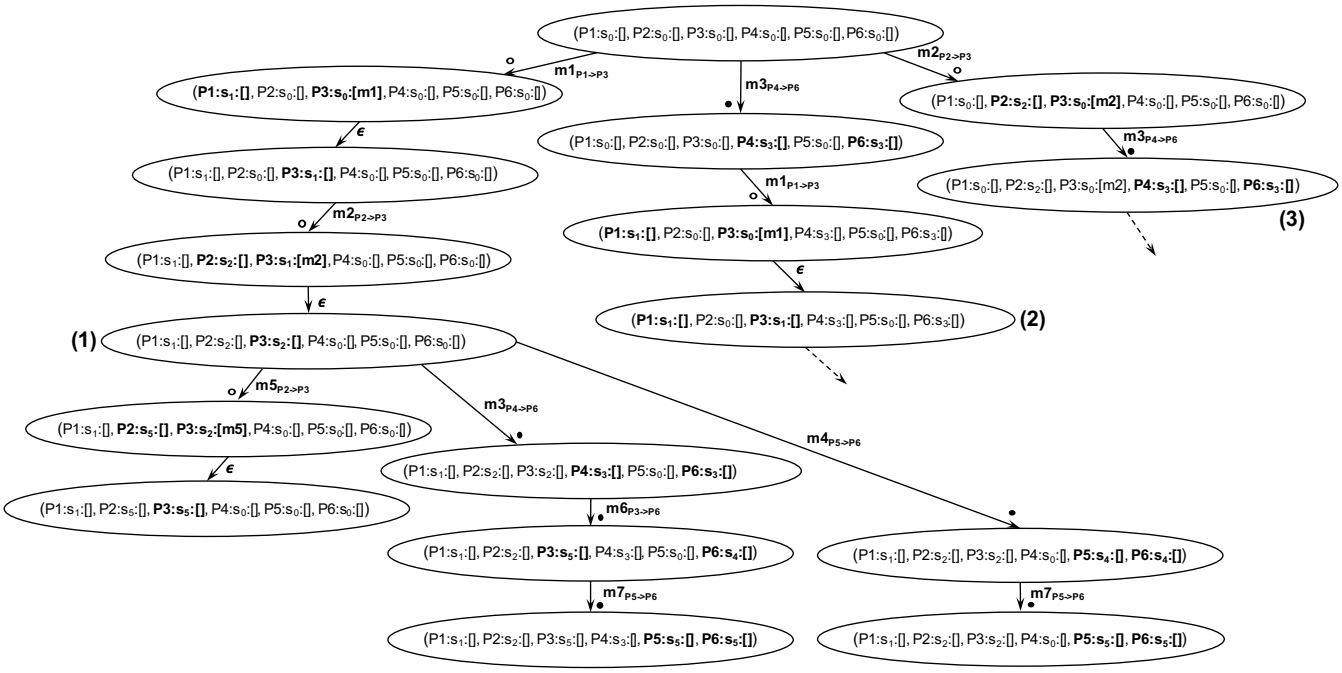

Fig. 9. An excerpt of the hybrid system composing the participants $P1$ to $P6$ (1-bounded queues)

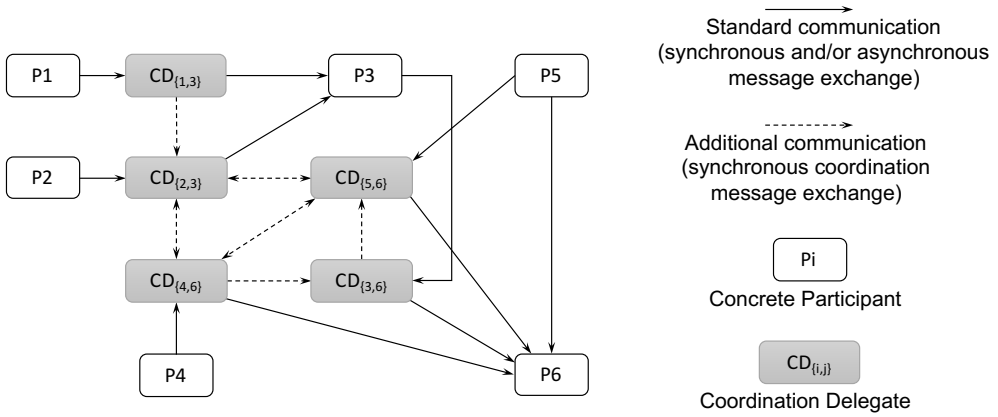

Fig. 10. Architectural style (a sample instance of)

IV. Related Work

The work described in this paper is related to approaches and tools for automated choreography realization.

In [3], the authors propose an approach to enforce synchronizability and realizability of a choreography. The method implementing the approach is able to automatically generate monitors, which act as local controllers interacting with their peers and the rest of the system in order to make the peers respect the choreography specification. Our notion of CD is "similar" to the notion of monitor used in [3], since CDs are able to interact with the choreography participants by also performing additional communication (i.e., the exchange of `Sync` messages) to exogenously coordinate the participants interaction so to fulfill the choreography specification. However, the two synthesis methods are different. In [3], the monitors are generated through an iterative process, automatically refining their behavior. They are first obtained by generating the set of peers via choreography projection. Then, the system synchronizability and realizability is automatically checked by using equivalence checking. If one of these properties is violated, the method exploits the generated counterexample to augment the monitors with a new synchronization message. Our synthesis method automatically generates CDs out of the choreography specification by considering that the selected concrete participants are (language) equivalent to the choreography abstract participants. The notion of realizability that we use in this paper is the same as the one used in [3] in that both works leverage results on choreography realizability and its decidability that are given in [16], [17].

In [4], the authors address the realizability problem based

on a priori verification techniques, using refinement and proof-based formal methods. They consider asynchronous systems where peers communicate via possibly unbounded FIFO buffers. The obtained asynchronous system is correct by construction, i.e., it realizes the choreography specification. With respect to our method and other methods discussed in this section, this method is more scalable in terms of number of involved peers and exchanged messages. However, our approach focuses on realizing a choreography specification by reusing third-party peers (possibly black-box), rather than generating the correct peers from scratch. This is why we cannot avoid to deal with exogenous coordination by means of additional software entities such as the CDs.

The approach in [5] checks the conformance between the choreography specification and the composition of participant implementations. The described framework can model and analyze compositions in which the interactions can be asynchronous and the messages can be stored in unbounded queues and reordered if needed. Following this line of research, the authors provided a hierarchy of realizability notions that forms the basis for a more flexible analysis regarding classic realizability checks [5]. In [6], the authors identify a class of systems where choreography conformance can be efficiently checked even in the presence of asynchronous communication. This is done by checking choreography synchronizability.

VerChor is a framework for choreography design and verification [1]. The framework checks a set of key properties that choreographies must respect for ensuring correctness of the system under development, by using verification techniques and tools for choreography analysis. The authors focuses on asynchronous communication semantics, that is, peers involved in the distributed system exchange messages via FIFO buffers. Although, in its current stage, MULTISYNTH STUDIO provides support to realize service choreographies only, differently to VerChor, it has been conceived and engineered to support the analysis and synthesis of reuse-based software, which is not limited to service choreographies.

The ASTRO toolset supports automated composition of Web services and the monitoring of their execution [7]. ASTRO deals with centralized orchestration-based processes rather than fully decentralized choreography-based ones.

In [8], the authors present a unified programming framework for developing choreographies that are correct by construction in the sense that, e.g., they ensure deadlock freedom and communication safety.

V. CONCLUSIONS AND FUTURE WORK

This paper presented the preliminary version of the MULTISYNTH STUDIO, a multipurpose framework for software integration synthesis. The studio offers extension mechanisms so to include other kind of integration synthesis methodologies. Our mid-term goal is to provide both researchers and practitioners with an easily accessible web-based environment that integrates different kinds of integrator synthesis approaches.

ACKNOWLEDGMENT

This research work has been supported by (i) the European Union's H2020 Programme under grant agreement number 644178 (project CHOReVOLUTION - Automated Synthesis of Dynamic and Secured Choreographies for the Future Internet), (ii) the Ministry of Economy and Finance, Cipe resolution n. 135/2012 (project INCIPICT - INnovating CIty Planning through Information and Communication Technologies), and (iii) the GAUSS national research project, which has been funded by the MIUR under the PRIN 2015 program (Contract 2015KWREMX).

REFERENCES

[1] M. Güdemann, P. Poizat, G. Salaün, and L. Ye, "Verchor: A framework for the design and verification of choreographies," *IEEE Transaction on Services Computing*, vol. 9, no. 4, pp. 647–660, 2016.

[2] M. Autili, P. Inverardi, and M. Tivoli, "Automated synthesis of service choreographies," *IEEE Software*, vol. 32, no. 1, pp. 50–57, 2015.

[3] M. Güdemann, G. Salaün, and M. Ouederni, "Counterexample guided synthesis of monitors for realizability enforcement," in *Automated Technology for Verification and Analysis*, ser. LNCS, S. Chakraborty and M. Mukund, Eds., 2012, pp. 238–253.

[4] Z. Farah, Y. Ait-Ameur, M. Ouederni, and K. Tari, "A correct-by-construction model for asynchronously communicating systems," *Int. Journal on Software Tools for Technology Transfer*, 2016.

[5] R. Kazhamiakin and M. Pistore, "Choreography conformance analysis: Asynchronous communications and information alignment," in *Web Services and Formal Methods*, ser. LNCS, 2006, vol. 4184.

[6] S. Basu and T. Bultan, "Choreography conformance via synchronizability," in *20th WWW*, 2011.

[7] M. Trainotti, M. Pistore, G. Calabrese, G. Zacco, G. Lucchese, F. Barbon, P. Bertoli, and P. Traverso, "ASTRO: supporting composition and execution of web services," in *3rd Int. Conf. on Service-Oriented Computing ICSOC*, 2005.

[8] M. Carbone and F. Montesi, "Deadlock-freedom-by-design: multiparty asynchronous global programming," in *40th ACM SIGPLAN-SIGACT POPL*, 2013, pp. 263–274.

[9] R. Calinescu, M. Autili, J. Cámara, A. D. Marco, S. Gerasimou, P. Inverardi, A. Perucci, N. Jansen, J. Katoen, M. Z. Kwiatkowska, O. J. Mengshoel, R. Spalazzese, and M. Tivoli, "Synthesis and verification of self-aware computing systems," in *Self-Aware Computing Systems*, 2017, pp. 337–373.

[10] M. Autili, P. Inverardi, F. Mignosi, R. Spalazzese, and M. Tivoli, "Automated synthesis of application-layer connectors from automata-based specifications," in *9th Int. Conf. on Language and Automata Theory and Applications LATA*, 2015, pp. 3–24.

[11] M. Autili, A. D. Salle, A. Perucci, and M. Tivoli, "On the automated synthesis of enterprise integration patterns to adapt choreography-based distributed systems," in *14th Int. Workshop on Foundations of Coordination Languages and Self-Adaptive Systems FOCLASA*, 2015.

[12] M. Autili, A. D. Salle, and M. Tivoli, "Synthesis of resilient choreographies," in *5th Int. Workshop on Software Engineering for Resilient Systems SERENE*, 2013, pp. 94–108.

[13] M. Autili, D. D. Ruscio, A. D. Salle, P. Inverardi, and M. Tivoli, "A model-based synthesis process for choreography realizability enforcement," in *16th Int. Conf. on Fundamental Approaches to Software Engineering FASE*, 2013, pp. 37–52.

[14] M. Autili, L. Mostarda, A. Navarra, and M. Tivoli, "Synthesis of decentralized and concurrent adaptors for correctly assembling distributed component-based systems," *Journal of Systems and Software*, vol. 81, no. 12, pp. 2210–2236, 2008.

[15] R. F. Paige, P. J. Brooke, and J. S. Ostroff, "Metamodel-based model conformance and multiview consistency checking," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 3, p. 11, 2007.

[16] S. Basu, T. Bultan, and M. Ouederni, "Deciding choreography realizability," in *39th ACM SIGPLAN-SIGACT POPL*, 2012, pp. 191–202.

[17] S. Basu and T. Bultan, "Automated choreography repair," in *19th Int. Conf. on Fundamental Approaches to Software Engineering FASE*, 2016, pp. 13–30.