Automated Component-Based Configuration: Promises and Fallacies

Sander van Splunter* Niek Wijngaards* Frances Brazier*

Debbie Richards†

*Department of Computer Science
Vrije Universiteit Amsterdam, The Netherlands
{svsplun,niek,frances}@cs.vu.nl

†Department of Computing, Macquarie University,
Sydney, Australia
richards@ics.mq.edu.au

Abstract

Re-use of software components is standard practice in software design and development in which humans play an important role. In many dynamic environments, however, (semi-)automated configuration of systems, is warranted. This paper examines three such domains: Agent Factories, Web service configuration and general software composition. The differences and similarities between these approaches, and the progress that is being made are discussed.

1 Introduction

Re-use of software components is part of many approaches to software design and development ((e.g.,Biggerstaff and Perlis (1997))). Most approaches assign an important role to human developers. In dynamic environments (semi-)automated configuration of systems can reduce, or even eliminate, the human effort required. In dynamic environments (semi-)automated configuration of systems from reusable components, is warranted. This paper examines three such domains: Agent Factories, Web service configuration and general software composition. The differences and similarities between these approaches, and the progress that is being made, are explicitly addressed.

Agents are active entities in dynamic, changing environments supported by the Internet. There are different ways for agents to adapt to such changing environments (e.g., see Splunter et al. (2003)). One is the Agent Factory approach in which first a need for change is identified and then agents are adapted. Another is the evolutionary approach in which agents continually adapt to their environment through implicit learning. The first mandates an understanding of the structure of an agent, and the components involved. The second mandates an understanding of the parameters involved in learning. This paper examines the first approach.

The Internet provides infrastructure to host agents and mobile processes. It also provides the infrastructure needed for both agents and humans to access Web services. In many business chains a number of Web services play a role. Web services need to be combined - configured. Availability of Web services, for example, is often crucial. If a particular Web service is not accessible an alternative service or combination of services needs to be found almost instantaneously. Automated configuration, although not currently acquired, is being actively pursued.

Within component based software engineering approaches automatic configuration is not often persued, but, when it is, it focuses mostly on the intial design. It, thus, provides the basis for both agent and Web service configuration.

This paper is organised as follows. Section 2 briefly introduces and analyses three Agent Factories. Section 3 introduces and analyses Web service configuration. Section 4 introduces and analyses one approaches to component-based configuration from a software engineering perspective. Section 5 compares these three overall approaches to component-based configuration by focussing on strengths and weaknesses on a number of aspects, related to the end-products, reusable components, and the configuration process. Section 6 concludes this paper with directions for future research.

2 Agent Factories

Agent Factories are either services or toolkits for (semi-)automated creation and (optionally) adaptation of software agents. These environments are strongly related to methodologies for agent application development. They include support for agent modelling (e.g. AUML), generic agent models (e.g by DESIRE, ZEUS, or InteR-Rap) and prototype generation. Examples of prototyping environments include the ZEUS toolkit (Nwana et al. (1998)), LEAP (Berger et al. (2001)), the (Dutch) Agent Factory (Brazier and Wijngaards (2001)), the (Irish) Agent Factory (Collier and O'Hare (1999)), and the (Italian) Agent Factory (Cossentino et al. (2003)). This paper

refers to the last three as the Dutch, Irish, and Italian agent factoriesand describes them in more detail in this paper.

In general, agent factories produce software agents, which are executed in the context of specific agent plaforms.

Dutch Agent Factory. This approach focuses on automation of the creation and adaptation of compositional agents. The nature of the components, of which an agent is composed, is graybox. These components provide mechanisms to implement an agent's processes, knowledge & information and control. Component composition is regulated by explicitly defined 'open slots' in components & templates based on generic agent models. Components are defined at two levels of abstraction: conceptual and operational. Minimal ontologies are used for annotation of components and interfaces, without adhering to standard ontologies or languages. Repositories for building blocks have not yet been further researched; only local re-use is supported.

The configuration process itself is automated, and includes reasoning on, and modification of, the requirements on the desired configuration. Configuration creation is automated, based on a generic model of design theory: the Generic Design Model (Brazier and Wijngaards (2002)). Retrieval and assembly are modelled as separate processes. Adaptation is approached as redesign, supported by the Generic Design Model. Component retrieval has not yet been automated. Assembly has been automated, partially being incorporated in the operational architecture used for the agents. Execution of configured agents is done by means of an agent platform. Agent platforms for which agents have been configured are AgentScape, FIPA-OS, and JADE.

Italian Agent Factory. The Italian Agent Factory is based on a toolkit that supports the multi-agent system design methodology PASSI (Cossentino and Potts (2002)).

The nature of the components is graybox. Components are configured and structured for five different configuration perspectives used within PASSI: knowledge, social, computer, architectural, and resource. The first two can be related to the conceptual level, and the remaining three to the operational level. Conceptual components are based on roles and tasks, operationally software component are used for modelling. Component composition is done by pattern merging on the conceptual level, on the operational level code is reused linked to the conceptual patterns. The interfaces of the software components are locally developed and defined.

The configuration process results in a skeleton of code that needs to be completed by a human programmer. Annotation of the different patterns is done using AUML, state charts, and activity diagrams. Annotation for human desigenrs is supported by the Rational Rose tool. The creation of an extensive repository of patterns is still an open research issue. The re-use community of the patterns is limited to the Italian Agent Factory.

The Italian Agent Factory's configuration creation process is semi-automated as a support tool. Configuration adaptation does not seem to be explicitly supported. Pattern retrieval seems to be mainly done by a human designer. The assembly of components is partially automated: a skeleton with partial completed code is the software product of the Italian Agent Factory. The execution is by means of a FIPA-compliant agent platform. The agents produced adhere to the interface standards set by FIPA for agent platforms.

Irish Agent Factory. The Irish Agent Factory is developed as a complete system to enable Agent Oriented Software Engineering, complete with a formal theory on agent commitments, agent programming language, and a run-time environment. This paper focuses on the tool to create agents.

The Irish software components are graybox, all programmed and developed within the same methodology. The conceptual components are modeled as roles and tasks, time with a focus on the use of commitments. The components' interfaces are developed and defined locally. Conceptually component composition is done by configuring a set of actuators and perceptors. Operationally agents are created by module-based development. Different default sets of actuators, perceptors, and modules are offered to support default implemnetations of different classes of agents. Annotation is application specific in the Irish Agent Factory's own high level programming language (AF-APL) with the addition of behaviour diagrams. Repositories of components have not been developed. Due to the specific theory, programming language, and run-time environment the reuse community consists only of the Irish Agent Factory.

Configuration creation is primarily done by a human designer, where the toolkit mainly acts as a smart interface. A number of default configurations is offered that can be used for initial configuration. Component retrieval is limited, a lookup service is associated which can (partial) retrieve designs via design identifiers. Assembly is done by either retrieving pre-fabricated configurations, or semi-automated by the toolkit. Execution is by means of the agent platform associated with the Irish Agent Factory.

Discussion. In the three Agent Factories discussed above agents are developed on the basis of instantiated 'patterns' (e.g. 'generic models'), or combinations of agent-components. The Italian and Irish agent factories focus on semi-automated generation; automated generation of agents is demonstrated by the Dutch agent factory. Automated adaptation of previously configured agents is only achieved by the Dutch Agent Factory. Nevertheless, all three Agent Factories pave the road for component-based agent *adaptation*.

All these Agent Factories produce agents for agent platforms that are FIPA compliant. Each agent factory uses its own approach to define components, interfaces, and annotation. All agent factories provide mechanisms related to process, data/information & knowledge, and control within software agents. All agent factories distinguish conceptual and operational levels of configuration. At the conceptual level, the Italian agent factory merges its components (patterns), while the Dutch and Irish agent factories combine components. All agent factories combine components at the operational level.

All agent factories are rather small-scale, both in the number of annotated components provided and associated (re-use) community. Annotation of components is often not explicitly supported, but sometimes implicit in the development of components (i.e. the Irish Agent Factory.

Not all agent factories use standard Software Engineering modelling support for creating agents. The Italians use AUML, state charts, activity diagrams, and extended Rational Rose. The Irish have explored the option of extending their methodology with AUML. The Dutch Agent Factory lacks standard SE modelling support technologies.

3 Web Service Configuration

The Stencil Group[1] defines web services as "loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols". Web services are related to the Semantic Web, in which data is defined and linked to make it accessible and interpretable for automated systems.

A configuration of Web services describes which Web services combined,control, and the information exchange. A configuration of Web services involves multiple processes on different hosts as individual Web services are offered and hosted by different parties. In contrast to agents, the behaviour of both a configuration and of single Web services is purely reactive and static. In general, two perspectives on Web service configuration can be identified (Srivastava and Koehler (2003)): a more syntactic-oriented Business Process composition and a more Semantic-Oriented service composition. The Business Process perspective models Web services as business processes, without attaching detailed semantics, and uses standardised technologies to describe Web services such as WSDL, SOAP, and UDDI. Web service configurations are described by orchestration languages such as XLANG, WFSL, BPEL4WS, or WSMF. This perspective is successful in human-supported discovery, composition, and monitoring of Web services. The Semantic-Oriented perspective extends the Business Process perspective by identifying the need of explicit semantics to enable the automation these tasks. Standard languages (e.g. OWL) and ontologies (e.g., OWL-S (formerly DAML-S)) are available for semantically annotating Web services. This paper focuses on the two Semantic-Oriented approach discussed below.

[1] http://www.stencilgroup.com/ideas_scope_200106wsdefined.html

eFlow. eFlow (Casati et al. (2000)), developed by the Hewlett-Packard Company, is oriented to adaptation of workflow models of composite Web services. Web services are treated as blackbox components. The component structure is based on workflows: activities with a data flow and control flow are modelled. The component interfaces are based on standard Internet protocols, and are described in WSDL. Component composition is regulated using a self-defined model, in which workflow concepts have been reused and extended. An XML specification and a DTD to constrain syntax are used for annotation. The component availability is low, due to the specific XML annotation.

eFlows main consideration is the focus on adaptation. Adaptations is performed in the context of monitoring Web service configuration, to minimize or eliminate human intervention. Component retrieval is done by brokers using centralised repositories, and the execution and monoring is performed by an eFlow engine.

Cardoso Cardoso and Sheth (2002) focus on the integration of new Web services in existing workflows. Web services are blackbox components. The component structure is based on modelling activities with a data flow and control flow, extended with annotations on the Quality of Service (QoS). Component composition is based on workflow integration, and supported by abstract Service Templates. For annotation DAML-S has been used in examples. The DAML-S Profile ontology has been extended to include more details on the QoS. This approach has a prototype with a self-defined local repository and discovery service, though usage of UDDI is also considered. The reuse community of the components is larger (the additional QoS attributes are only extensions to existing standards, not rendering components incompatible with other approaches). Component availability, including QoS annotations, is small.

In the configuration process of Cardoso's approach configuration adaptation is semi-automated. The human designer is supported in the creation and refinement of abstract Service Templates. This approach is not specifically targeted to the creation of Web service configurations from scratch. Component retrieval is semi-automated: based on an abstract Service Template possible Web services for refinements are retrieved based on aspects of QoS, similarity of textual descriptions or names, and semantic similarity of inputs and outputs. The assembly and execution a configuration is not clear.

Discussion. Within Web services configuration the services are blackbox components with standardised inter-

faces. The component interface standards are based on standard Internet protocols and described using WSDL and SOAP: generally accepted standards. The component structure is mostly process-based: in the workflow perspective it consists of activities with a control and data flow.

The components themselves are annotated in standard annotation languages as WSDL and OWL. The latter language is mostly used by semantic-oriented matching approaches, for which standard ontologies for Web services are available in OWL-S. Annotating Web services is an inherent part of the Web service development process. Unfortunately, most approaches do not focus on creating rich domain ontologies to be reused when describing other services. A large number of Web services is available where annotations are limited to only the WSDL descriptions. A smaller number of components is available with semantic rich OWL-S descriptions

The use of globally shared repositories support a global reuse community and widely available Web services. The communities involved in the semantic Web are still growing, which may positively influence the availability of well-annotated re-usable Web services.

As Web services, by their very nature, can appear, change, or disappear while being used in compositions, the need for automated adaptation is recognised, and progress is made. Most approaches to automated configuration are still in development; configuration creation is often approached as a (simple) planning problem. Component retrieval is semi-automated, i.e. UDDI is used as a repository that can be queried, but human intervention is still required to determine whether the resulting Web service are useable. The Semantic-Oriented perspective has extended (e.g. MatchMaker by Paolucci et al. (2002)) UDDI to handle further automated semantic querying, by including approximate answers. Execution of Web services configurations is done by workflow execution engines, e.g., BPWS4J, and the BPEL Orchestration Server. For execution of DAML-S descriptions research is still evolving (e.g. see Gaio et al. (2003)), as the research area is still young.

4 Component-Based Software Engineering

Component-based software engineering focuses, in general, on developing components (e.g., CORBA, Java Beans, .NET). while software composition is often only supported by tools, not automated, although exceptions are present. Examples of modelling support are UML [2], or tools such as Rational Rose [3]. In this paper Quasar(de Bruin and van Vliet (2003)), a semi-automated approach to software composition based on feature composition, is discussed.

[2] http://www.uml.org

[3] http://www.rational.com

Quasar. Quasar is a tool to support top-down composition of software architectures. In this approach, an architecture is derived to fulfill a of quality concerns. A Quasar specific feature-solution graph is used to connect quality requirements to solution fragments at the architectural level: a form of composition knowledge. An architecture is derived by systematically composing solution fragments. Both functional and non-functional requirements are addressed.

The reusable components, in this approach design solutions, are often patterns but may also be individual software components. Design solutions are represented by use case maps (UCM) whereby both behavioural and structural aspects are expressed. Components are distinguished from sockets and stubs, with which component composition is regulated, via a refinement process. Quasar uses BCOOPL(de Bruin (2003)) to specify interface definitions, including pre- and post-conditions and -processing via pre- and post-stubs. Composition templates support composition of design solutions.

Annotation of design solutions are encoded in feature-solution graphs. These graph are not automatically generated. A number of prototype components are available. The configuration creation process is semi-automated: architectures are derived iteratively by first generating a reference architecture, focussing on functional requirements followed by non-functional requirements. The choices of which requirements to focus on, is left to the human designer. This process may include backtracking to resolve conflicting requirements.

Discussion. Component-based software configuration is a broad field. Automated approaches in component-based software engineering are often very domain-specific (comparable with agent-configuration and Web service configuration). In general, components are well-defined structures, for which composition is explicitly regulated by defining 'hooks'. Both components and hooks are well-defined on a syntactical level, sometimes involving semantic annotations (by associating features to solutions in Quasar). Only small sets of annotated components are available, and the more general an approach, the more support is provided for large-scale repositories. Simple, exact-matching, retrieval is often provided. Configuration of software components is almost never fully automated. This is due, in general, to software components being difficult to reuse as their domain specificity conflicts with reuse genericity (Sametinger (1997)). Assembly of software compositions is, in general, supported.

5 Comparative Analysis

Agents, services and general compositional software have different characteristics, but also commonalities. The comparison in this section is structured by focussing on

four aspects: component definition, component annotation, component availability, and configuration process.

Component definition. Although all approaches explicitly define components, the Web service approaches employ standards, used by a large community. The component definition, however, is a blackbox approach, without providing hooks for composition: composition requires new configuration languages. In the Agent Factories and component-based software engineering components are graybox, providing hooks for composition. All approaches distinguish conceptual and operational levels in their component definitions.

Web services are intended to be a globally reused, in an open domain. The Web service community builds on the existing (Web-)protocols, e.g. SOAP, HTTP, while the agent community has inter-agent protocols, but have no standards for the interfaces of the components of which agents themselves are composed. The software composition approach provide no restrictions on domain and/or interfaces, yet automated configuration is limited to specific domains of application.

Component annotation. The adoption of standards for annotation facilitates the development of automated configuration processes. Only the Web service community has emerging computer-interpretable annotation standards, the other approaches do not as yet. Human-understandable annotations of software engineering, like UML, are widely applied and accepted (broader than the Web service annotation standards). The annotations of the Web service communities do not extend these SE standards. Within the Agent modelling community the SE annotations are more generally accepted (e.g. AUML).

Component availability. Automated configuration of software depends on the availability of configurable software components: a critical mass is needed for automated configuration to be successful. Annotated Web services are becoming more widely available, due to a large supportive community and the creation of semantic descriptions being part of the development process of Web services. Agents, however, are often developed without a focus on re-usability of agent-components, leading to scarce availability of annotated agent-components. Software components are being developed in large quantities, around the world, but are usually not geared for automated reuse, lacking computer-interpretable annotations.

Configuration processes. Fully automated configuration is not widely supported. All communities research automated adaptation, whereby the agent community often focuses on learning algorithms without structural adaptation. The Web service community focuses on adaptation, mostly studied in the context of business-processes. Component-based software engineering has traditionally had its main focus semi-automated adaptation, but is moving towards automated adaptation (e.g., see also the developments in self-managing, and self-healing systems).

Discovery and retrieval of software components is important. Annotation of components will facilitate their automatic discovery and retrieval. The Semantic Web community experiments with public services, and uses reasoning about the annotations. Within the Agent Factories retrieval of re-usable components is not extensively studied, and no large repositories are publicly available. For software-composition communities repositories are available, however mainly for computer-supported discovery and retrieval. For automated discovery and retrieval the repositories are often limited in size and non-public.

6 Discussion & Future Work

The comparison is limited, discusses the issues involved in each of the approaches. Internet applications require flexibility to accomodate changes in their environment. As manual adaptation is not pragmatic when multitudes of agents and Web services are in use, automated adaptation becomes a necessity. Unfortunately, the current state of the art, even in software engineering, does not include fully automated component-based adapation. Current research focuses on component-based configuration of agents, Web services, and software composition: a precondition for adaptation. Progress has been made in automation of component-based configuration.

Automated component-based configuration of, e.g., software agents, entails a thorough understanding of both configuration processes, and the components to be configured. The agent community has made the most progress in automation of the configuration process. The Web service community has made the most progress in development and annotation and in the discovery and retrieval of these components. The software composition community has made the most progress in structuring and modelling components, and on supporting the human designer. Interdisciplinary research may be most fruitful, when (1) combining configuration-expertise with annotation-expertise, (2) generalising and standardising reusable, configurable, components, and (3) when the current structuring and modelling practices of SE are used. Once automated component-based configuration has proven to be feasible, research can focus on automated component-based adaptation as a new challenge.

Acknowledgements

The authors wish to thank Marta Sabou for her work on Web service configuration. The authors are also grateful to the support provided by Stichting NLnet, http://www.nlnet.nl/.

References

N. Berger, B. Bauer, and M. Watzke. A scalable agent infrastructure. In *2nd Workshop on Infrastructure for Agents, MAS and Scalable MAS. Autonomous Agents.01, Montreal*, 2001.

T. Biggerstaff and A. Perlis, editors. *Software Reusability: Concepts and models*, volume 1. ACM Press, New York, 1997.

F.M.T. Brazier and N.J.E. Wijngaards. Automated servicing of agents. *AISB Journal*, 1(1):5–20, 2001. Special Issue on Agent Technology.

F.M.T. Brazier and N.J.E. Wijngaards. Automated (re-)design of software agents. In J.S. Gero, editor, *Proceedings of the Artificial Intelligence in Design Conference 2002*, pages 503–520. Kluwer Academic Publishers, 2002.

J. Cardoso and A. Sheth. Semantic e-workflow composition. Technical report, LSDIS Lab, Computer Science Department, University of Georgia, 2002. http://chief.cs.uga.edu/~jam/webwork/geneflow/papers/CS02-_20Composition_20-_20TR.pdf.

F. Casati, S. Ilnicki, and L. Jin. Adaptive and dynamic service composition in eflow. HP Technical Report HPL-2000-39, HP, 2000. http://www.hpl.hp.com/techreports/2000/HPL-2000-39.pdf.

R.W. Collier and G.M.P. O'Hare. Agent factory: A revised agent prototyping environment. In *10th Irish Conference on Artificial Intelligence & Cognitive Science*, University College Cork, Ireland, 1999.

M. Cossentino, P. Burrafato, S. Lombardo, and L. Sabatucci. Introducing pattern reuse in the design of multi-agent systems. In R. Kowalczyk, J.P. Muller, H. Tianfield, and R. Unland, editors, *Agent Technologies, Infrastructures, Tools, and Applications for E-Services: NODe 2002 Agent-Related Workshops, Erfurt, Germany, October 7-10, 2002.*, volume 2592 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 107–120, 2003.

M. Cossentino and C. Potts. A case tool supported methodology for the design of multiagent systems. In *Proceedings of the 2002 International Conference on Software Engineering Research and Practice (SERP'02)*, Las Vegas, NV, USA, 2002.

H. de Bruin. Bcoopl: A language for controlling component interactions. *The Journal of Supercomputing*, 24 (2):131–139, 2003.

H. de Bruin and H. van Vliet. Quality-driven software architecture composition. *Journal of Systems and Software*, 66(3):269–284, 2003.

S. Gaio, A. Lopes, and L Botelho. From daml-s to executable code. In B. Burg, J. Dale, T. Finin, H. Nakashima, Padgham L., C. Sierra, and Willmott S., editors, *Agentcities: Challenges in Open Agent Environments*, pages 25–31. Springer-Verlag, 2003.

H.S. Nwana, D.T. Ndumu, and L.C. Lee. Zeus: An advanced tool-kit for engineering distributed multi-agent systems. *Applied AI*, 13(1/2):129–185, 1998.

M. Paolucci, T. Kawamura, T.R. Payne, and K.P. Sycara. Semantic matching of web services capabilities. In *Proceedings of the International Semantic Web Conference 2002*, pages 333–347, 2002.

J. Sametinger. *Software engineering with reusable components*. Springer Verlag, New York, 1997.

S. van Splunter, N.J.E. Wijngaards, and F.M.T. Brazier. Structuring agents for adaptation. In E. Alonso, D. Kudenko, and D. Kazakov, editors, *Adaptive Agents and Multi-Agent Systems*, volume 2636 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 174–186. Springer-Verlag Berlin, 2003.

B. Srivastava and J. Koehler. Web service composition - current solutions and open problems. In *ICAPS 2003 Workshop on Planning for Web Services*, pages 28–35, 2003.